Intergraph Smart 3D

Structure

3D Symbols Reference

Version 2016 (11.0)
November 2016

Copyright

Copyright © 2001-2016 Intergraph® Corporation. All Rights Reserved. Intergraph is part of **Hexagon**.

Including software, file formats, and audiovisual displays; may be used pursuant to applicable software license agreement; contains confidential and proprietary information of Intergraph and/or third parties which is protected by copyright law, trade secret law, and international treaty, and may not be provided or otherwise made available without proper authorization from Intergraph Corporation.

Portions of this software are owned by Spatial Corp. © 1986-2016. All Rights Reserved.

Portions of the user interface are copyright © 2012-2016 Telerik AD.

U.S. Government Restricted Rights Legend

Use, duplication, or disclosure by the government is subject to restrictions as set forth below. For civilian agencies: This was developed at private expense and is "restricted computer software" submitted with restricted rights in accordance with subparagraphs (a) through (d) of the Commercial Computer Software - Restricted Rights clause at 52.227-19 of the Federal Acquisition Regulations ("FAR") and its successors, and is unpublished and all rights are reserved under the copyright laws of the United States. For units of the Department of Defense ("DoD"): This is "commercial computer software" as defined at DFARS 252.227-7014 and the rights of the Government are as specified at DFARS 227.7202-3.

Unpublished - rights reserved under the copyright laws of the United States.

Intergraph Corporation
305 Intergraph Way
Madison, AL 35758

Documentation

Documentation shall mean, whether in electronic or printed form, User's Guides, Installation Guides, Reference Guides, Administrator's Guides, Customization Guides, Programmer's Guides, Configuration Guides and Help Guides delivered with a particular software product.

Other Documentation

Other Documentation shall mean, whether in electronic or printed form and delivered with software or on Intergraph Smart Support, SharePoint, or box.net, any documentation related to work processes, workflows, and best practices that is provided by Intergraph as guidance for using a software product.

Terms of Use

a. Use of a software product and Documentation is subject to the End User License Agreement ("EULA") delivered with the software product unless the Licensee has a valid signed license for this software product with Intergraph Corporation. If the Licensee has a valid signed license for this software product with Intergraph Corporation, the valid signed license shall take precedence and govern the use of this software product and Documentation. Subject to the terms contained within the applicable license agreement, Intergraph Corporation gives Licensee permission to print a reasonable number of copies of the Documentation as defined in the applicable license agreement and delivered with the software product for Licensee's internal, non-commercial use. The Documentation may not be printed for resale or redistribution.

b. For use of Documentation or Other Documentation where end user does not receive a EULA or does not have a valid license agreement with Intergraph, Intergraph grants the Licensee a non-exclusive license to use the Documentation or Other Documentation for Licensee's internal non-commercial use. Intergraph Corporation gives Licensee permission to print a reasonable number of copies of Other Documentation for Licensee's internal, non-commercial use. The Other Documentation may not be printed for resale or redistribution. This license contained in this subsection b) may be terminated at any time and for any reason by Intergraph Corporation by giving written notice to Licensee.

Disclaimer of Warranties

Except for any express warranties as may be stated in the EULA or separate license or separate terms and conditions, Intergraph Corporation disclaims any and all express or implied warranties including, but not limited to the implied warranties of merchantability and fitness for a particular purpose and nothing stated in, or implied by, this document or its contents shall be considered or deemed a modification or amendment of such disclaimer. Intergraph believes the information in this publication is accurate as of its publication date.

The information and the software discussed in this document are subject to change without notice and are subject to applicable technical product descriptions. Intergraph Corporation is not responsible for any error that may appear in this document.

The software, Documentation and Other Documentation discussed in this document are furnished under a license and may be used or copied only in accordance with the terms of this license. THE USER OF THE SOFTWARE IS EXPECTED TO MAKE THE FINAL EVALUATION AS TO THE USEFULNESS OF THE SOFTWARE IN HIS OWN ENVIRONMENT.

Intergraph is not responsible for the accuracy of delivered data including, but not limited to, catalog, reference and symbol data. Users should verify for themselves that the data is accurate and suitable for their project work.

Limitation of Damages

IN NO EVENT WILL INTERGRAPH CORPORATION BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL INCIDENTAL, SPECIAL, OR PUNITIVE DAMAGES, INCLUDING BUT NOT LIMITED TO, LOSS OF USE OR PRODUCTION, LOSS OF REVENUE OR PROFIT, LOSS OF DATA, OR CLAIMS OF THIRD PARTIES, EVEN IF INTERGRAPH CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

UNDER NO CIRCUMSTANCES SHALL INTERGRAPH CORPORATION'S LIABILITY EXCEED THE AMOUNT THAT INTERGRAPH CORPORATION HAS BEEN PAID BY LICENSEE UNDER THIS AGREEMENT AT THE TIME THE CLAIM IS MADE. EXCEPT WHERE PROHIBITED BY APPLICABLE LAW, NO CLAIM, REGARDLESS OF FORM, ARISING OUT OF OR IN CONNECTION WITH THE SUBJECT MATTER OF THIS DOCUMENT MAY BE BROUGHT BY LICENSEE MORE THAN TWO (2) YEARS AFTER THE EVENT GIVING RISE TO THE CAUSE OF ACTION HAS OCCURRED.

IF UNDER THE LAW RULED APPLICABLE ANY PART OF THIS SECTION IS INVALID, THEN INTERGRAPH LIMITS ITS LIABILITY TO THE MAXIMUM EXTENT ALLOWED BY SAID LAW.

Export Controls

Intergraph Corporation's software products and any third-party Software Products obtained from Intergraph Corporation, its subsidiaries, or distributors (including any Documentation, Other Documentation or technical data related to these products) are subject to the export control laws and regulations of the United States. Diversion contrary to U.S. law is prohibited. These Software Products, and the direct product thereof, must not be exported or re-exported, directly or indirectly (including via remote access) under the following circumstances:

a. To Cuba, Iran, North Korea, Sudan, or Syria, or any national of these countries.

b. To any person or entity listed on any U.S. government denial list, including but not limited to, the U.S. Department of Commerce Denied Persons, Entities, and Unverified Lists, http://www.bis.doc.gov/complianceandenforcement/liststocheck.htm, the U.S. Department of Treasury Specially Designated Nationals List, http://www.treas.gov/offices/enforcement/ofac/, and the U.S. Department of State Debarred List, http://www.pmddtc.state.gov/compliance/debar.html.

c. To any entity when Licensee knows, or has reason to know, the end use of the Software Product is related to the design, development, production, or use of missiles, chemical, biological, or nuclear weapons, or other un-safeguarded or sensitive nuclear uses.

d. To any entity when Licensee knows, or has reason to know, that an illegal reshipment will take place.

Any questions regarding export or re-export of these Software Products should be addressed to Intergraph Corporation's Export Compliance Department, Huntsville, Alabama 35894, USA.

Trademarks

Intergraph, the Intergraph logo, PDS, SmartPlant, FrameWorks, I-Sketch, SmartMarine, IntelliShip, ISOGEN, SmartSketch, SPOOLGEN, SupportManager, SupportModeler, Sapphire, and Intergraph Smart are trademarks or registered trademarks of Intergraph Corporation or its subsidiaries in the United States and other countries. Hexagon and the Hexagon logo are registered trademarks of Hexagon AB or its subsidiaries. Microsoft and Windows are registered trademarks of Microsoft Corporation. ACIS is a registered trademark of SPATIAL TECHNOLOGY, INC. Infragistics, Presentation Layer Framework, ActiveTreeView Ctrl, ProtoViewCtl, ActiveThreed Ctrl, ActiveListBar Ctrl, ActiveSplitter, ActiveToolbars Ctrl, ActiveToolbars Plus Ctrl, and ProtoView are trademarks of Infragistics, Inc. Incorporates portions of 2D DCM, 3D DCM, and HLM by Siemens Product Lifecycle Management Software III (GB) Ltd. All rights reserved. Gigasoft is a registered trademark, and ProEssentials a trademark of Gigasoft, Inc. VideoSoft and VXFlexGrid are either registered trademarks or trademarks of ComponentOne LLC 1991-2013, All rights reserved. Oracle, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Tribon is a trademark of AVEVA Group plc. Alma and act/cut are trademarks of the Alma company. Other brands and product names are trademarks of their respective owners.

Contents

Preface

This document is a guide for Intergraph Smart™ 3D symbols reference data. The purpose of this document is to describe how to create and customize the symbol reference data so that it fits your company or project.

For information about the specific reference data for each discipline, see the reference data guides available from the **Help > Printable Guides** command in the software.

Document Audience

This document is intended for advanced users who should:

- Have a good understanding of Microsoft® Office products, especially Microsoft Excel.
- Be familiar with Smart 3D database architecture and relational databases in general.
- Have a working knowledge of Solid Edge™ and Visual Basic® in order to create and modify three-dimensional symbols. For cross-sectional symbols, you should be familiar with Intergraph SmartSketch® or a similar product.

  NOTE Using Solid Edge to create and modify three-dimensional symbols is supported only for equipment symbols.

Related Documents

For more information about Smart 3D, please see the following documents:

Intergraph Smart™ 3D *Installation Guide*

Reference Data Guide

Documentation Comments

For the latest support information for this product, comments or suggestions about this documentation, and documentation updates for supported software versions, please visit *Intergraph Smart Support* (*https://smartsupport.intergraph.com*).

What's New in Structure Symbols

Version 2016 (11.0)

- Updated the information on using Solid Edge symbols in Smart 3D. For more information, see *Creating Symbols in Solid Edge* (on page 36). (P3 CP:237967)

SECTION 1

Symbols

Whether using one of the delivered symbols, or a custom symbol that you define yourself, symbols are a key building block used to create your model. There are two basic types of symbols that the software uses: 2-D and 3-D.

The 2-D symbols are used to represent structural member cross-sections, slots, collars and clips, brackets, and standard openings. You can use any of the defined cross sections or define your own custom cross-sections. For more information about 2-D symbols, refer to the *2D Symbols User's Guide*.

The 3-D symbols are used to represent equipment, hangers, HVAC components, piping components, and so forth in your model. There are hundreds of symbols that you can use as-is or customize to fit your needs. You can also create your own symbols. This document describes how to create symbols, incorporate them into your reference data, and describes the parameters of the delivered symbols.

In addition to the symbols delivered with the software, Intergraph provides symbols and symbol fixes on the *Intergraph Smart Support* (*https://smartsupport.intergraph.com*) web site. These symbols are available on the product page under **Downloads > Smart 3D > Content**.

In order to fully understand symbols, you need to learn a few terms:

- Symbol - A symbol is a custom business object that provides a symbolic representation of a set of graphics. It is possible for this set of graphics to look completely different in the different display aspects.
- Flavor - A flavor is the persistent cache of all the graphic objects displayed by a symbol. Each symbol visible in a session is just a symbolic representation (geometric transformation) of the graphics stored in a flavor.
- Symbol Definition - A symbol definition is the persistent template for all symbols in a database. It is the definition of the inputs, outputs, and options of all symbols created using this symbol definition.
- Flavor Manager - When many symbols use the same flavor, a flavor manager object is created to manage the relationships between the symbols, flavor, and symbol definition.
- Custom Component - A special symbol that has no flavor. Each custom component is a unique symbol containing its graphic objects.
- Outputs - Persistent objects that are created by the symbol when it calculates. The most common form of output is a graphic object, but output can be parameters.
- Inputs - Optional persistent objects used by a symbol to calculate its outputs.

See Also

Troubleshooting Symbols (on page 45)

2D Symbols

The 2D Symbols application is used to create 2-D symbols used to represent profile cross-sections, detailed parts, features, and end cuts in the Molded Forms and Structural Detailing tasks, and member cross-sections in the Structure task. The main purpose of 2D Symbols is to graphically create a flexible symbol definition so that it can be used to place different objects in a model. Two-dimensional symbols are delivered in the *[Product Folder]*\SharedContent\CrossSections folder.

You use 2D Symbols to create:

- The graphic representation or inputs of the symbol.
- Named symbol geometry, such as edge names used to orient the symbol in the 3-D environment and to constrain different types of symbols to each other.
- Parameters, such as driving dimensions.
- Geometric constraints (relationships) that specify which reference data parameters control which part of the symbol.
- Multiple representations, which can be selected in the model to control how the symbol is displayed.
- Additional auxiliary graphic objects to create and constrain symbols. These auxiliary objects do not become a part of symbol output geometry.

The utility also provides a dialog box for you to write the cross-section or profile into an Excel workbook, which you can bulk load into the catalog.

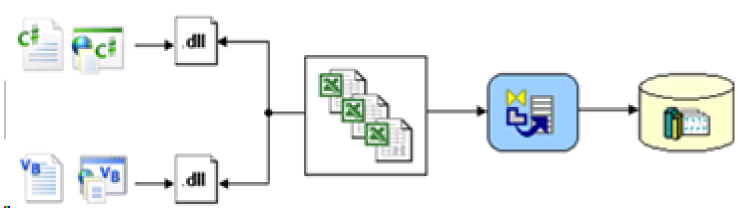
For more information, see the *2D Symbols User's Guide* available from the **Help > Printable Guides** command in the software.

3D Symbols

You can customize additional three-dimensional symbols for your company using .NET coding.

The following picture shows the types of symbols and corresponding file formats .NET symbols are in .vb or .cs format and will be delivered in .dll. The .dll symbols are registered on the computer used to host the SharedContent share.

The tabular data for the symbols resides in the excel workbooks. For example, you list the symbol name for the part on the part class sheet. You can use the Bulkload utility to load the excel data into the Catalog Database.



Each .NET 3-D symbol comprises source code (.csproj and .cs or vbproj and .vb) and a compiled file (.dll). The .dll files for the delivered 3-D symbols are located on the server computer at *[Product Folder]*\SharedContent\bin. This folder is shared to allow client computers

to access the symbols. You specify this folder when you bulk load reference data. If necessary, you can change the location when you bulk load a new catalog.

The 3-D symbol source code (.csproj and .cs or .vbproj or .vb) files are delivered during the Programming Resources Installation. For more information on installing the Programming Resources, refer to the *Smart 3D Installation Guide*.

To change a symbol, you must edit and then build the .NET code for the symbol. The new .dll can be added to the Custom Symbols folder in the SharedContent share. You also must edit the applicable bulk load workbook for the symbol, and bulk load the modified reference data into the Catalog database.

IMPORTANT When you add a new custom DLL to the *[Reference Data Folder]*\SharedContent\Custom Symbols folder, or when you edit an existing custom DLL, you must run the **Tools > Update Custom Symbol Configuration** command in Project Management.

The overall workflow for creating a part is as follows:

- Create or modify a .NET project.
- Compile to create a .dll.
- Create or modify an Excel workbook to create the part information. As an alternative to the workbooks, you can create part classes and part information in the Catalog task using the **Catalog > New > Class** command. Refer to the *Catalog User's Guide* for more information.
- Bulk load the workbook. You do not need to bulk load anything if you create your part classes in the Catalog task using the **Catalog > New > Class** command.
- Test the symbol in the software.

NOTE If you add new part classes after creating the Reports databases, you must re-create the Reports databases in order to report on the new part classes.

See Also
Creating .NET Symbols (on page 14)
Creating Symbols in Solid Edge (on page 36)

Defining Ports on Symbols

Most symbols have at least one port, which is a point on a part that connects to a routed item such as pipe or cable. A port consists of an attachment point and direction, a set of application properties, and a physical geometry depiction. A different class of port is required for each type of routing item. For example, piping requires one type of port, while cable requires another.

Defining Ports

You define ports when you create a symbol and define the geometry of a part. You can create three-dimensional symbols using .NET coding. In .NET, a function specifies the port type, name, attachment point, and attachment vector.

The software places the ports based on the information in the geometry definition file for the part and the reference data for the part. The geometry definition file defines the port type, name, attachment point, and attachment vector. The reference data for the specific part (item of the part class) defines the remainder of the property values for the port.

Modifying Ports

If you want to reposition a port on a part in the model, you must edit the geometry definition in .NET. You should do this task only before any occurrences of the part are placed in the model.

A port is related to the part to which it is attached. When you move the part, the port also moves. When you delete the part, the port is also deleted.

Providing a Graphical Preview

To make selecting and placing parts from the catalog easier, you can provide a preview graphic of the part. This graphic helps you to visually identify the correct part in the catalog for placement and should include any symbol dimensions that can be edited by you.

In the Catalog task, the **Preview** command on the **View** menu displays the preview graphic for the item. You can see preview graphics when you place items in the design tasks by clicking **Preview** on the Catalog browser from design tasks such as Equipment and Furnishings. In addition, some **Properties** dialog boxes in the design tasks have a button that allows you to see a preview of the selected item.

To add a preview graphic to the reference data, you must create a graphic file and store it in a shared symbol folder on a networked computer. For example, you can place the graphic file in *[Product Folder]*\SharedContent\Data, the default location installed during the Smart 3D Server setup.

You can define a preview graphic for a specific part, which overrides any preview graphics assigned to the part class. Any graphics created for individual parts must be stored in the same location as those defined for part classes.

To link the preview graphic to the part or part class:

1. Edit the Microsoft Excel workbook that contains the part class information.
2. In the **SymbolIcon** cell, type the path and preview graphic name.
3. Bulk load the workbook into the Catalog Database using the bulkload utility.

Graphic Recommendations

- The graphic must be a Windows Bitmap (.bmp) or a CompuServe Graphics Interchange (.gif) file. We recommend the .gif format because of the smaller file size.
- The graphic resolution should be 37 pixels per centimeter (94 pixels per inch).
- Use the lowest color depth possible without loss of image quality. Generally, this is 256 Colors (8 bit). However, some graphics can be dropped to 16 Colors (4 bit) or 2 Colors (1 bit) without loss of image quality.
- Use Verdana font with a font point size of 10 or 12 to place text in the graphic. We recommend the Verdana font because `1` (one), `I` (capital i), and l (lower case L) can be distinguished from one another in that font.
- Graphic dimensions should be as small as possible to allow you to have the graphic open while working with the software. The maximum graphic dimension that you should create is 974 X 718 (50 pixels less than the default screen resolution of 1024 X 768). The software does not limit the size of the graphic, so larger graphics can be used if your default screen resolution is higher.

Add a Preview Graphic to Parts using Bulkload

1. Create a graphic file (.bmp or .gif) in a graphics package.

   TIPS

   - The purpose of this graphic is to help you identify the correct part in the catalog. The graphic also can assist in identifying dimensions on a part.
   - You can create the graphic from a snapshot of a two-dimensional drawing or of the three-dimensional model. You also can draw the graphic freehand in a graphics package.
   - The graphic pixel limitation is about the size of your screen because the preview box in the Catalog task will automatically re-size around the graphic.

2. Save the graphic file in a shared symbol folder on the server. For example, you can place the graphic file in *[Product Folder]*\SharedContent\Data, the default location installed during the Smart 3D server setup.
3. Open the Excel workbook with the part class or part to which you want to add the preview graphic.
4. Select a part class sheet.

   TIP For example, if you want to add a preview graphic to the Pump class in the Equipment workbook, open **Equipment.xls** and select the **Pump** sheet.

5. In the **Definition** section on the sheet, add a column.
6. Type **SymbolIcon** at the top of the new column.
7. Below the **SymbolIcon** heading, type the name of the graphic file for the part class, such as **Pump.bmp**.
8. In the **Head/Start/End** section, type **SymbolIcon** for the column heading in the new column.
9. Type the name of a graphic file beneath the **SymbolIcon** heading in the **Head/Start/End** section.

   This graphic file defines the preview for the specific PART. The part graphic overrides the preview graphic for the PART CLASS.

   TIPS

   - If you want a part to have the same symbol file as the parent part class, type **NULL** beneath the **SymbolIcon** heading in the **Head/Start/End** section. Or, you can leave the cell blank.

- The following picture shows an Excel sheet that lists a symbol icon.

!	PartNumber should be unique [in the entire catalog]			
Definition	PartClassType	SymbolDefinition	SymbolIcon	Nozzle(1):Id
	EquipmentClass	Pump.PumpServices	SymbolIcons\Pump.bmp	Suction
Head	PartNumber	PartDescription	SymbolIcon	SymbolDefinition
Start				
	PUMP 001A	Centrifugal Pump		
	PUMP 001A_IMP	Centrifugal Pump		
	CPump002A8x6	Centrifugal Pump 1.5m^3/s, 8" suction, 6" discharge		
End				

10. Mark all of the rows that you modified with the letter **M**.
11. Bulkload the workbook in the **Add/Modify/Delete** mode. For more information about bulkloading, see *Bulk Load Database with Data* in the *Reference Data Guide*.

NOTES

- If you do not want to specify a preview graphic for a part class or part, do not add the **SymbolIcon** heading to the **Definition** or **Head/Start/End** sections. You do not have to specify a preview graphic for a part class or part.
- You can check the preview by starting the **Catalog** task, selecting the part or part class, and clicking **View > Preview**. You also can see the preview by selecting an item in the model and displaying the **Properties** dialog box for the item. Some **Properties** dialog boxes have a button that allows you to see a preview of the selected item.
- The software delivery includes preview symbols for several items. The delivery location for many of the preview symbols is *[Product Folder]*\SharedContent\Data on the server computer. If you want to add symbols, you must create the graphic and bulkload as described above.

See Also

Providing a Graphical Preview (on page 11)

SECTION 2

Creating .NET Symbols

Creating Smart 3D symbol content using the .NET 3D API provides the following benefits:

1. Access to the new and improved 3D API application objects and services.
2. Use of the 3D API from within the latest .NET framework.
3. Built-in support for running in future 64-bit applications.

To place a symbol in the 3D environment, you need graphical data (.NET symbol) and non-graphical data (Excel workbook containing specifications, rules, dimensional data, and so forth).

To create a symbol in the software, do the following:

1. Understand the geometry and use of the symbol.
2. Write .NET code for the geometry outputs to be created.
3. Sign the assembly with a strong name.
4. Bulkload the symbol data into the catalog database.

NOTE You do not need to manually grant the computer access to the .NET assemblies. The software automatically handles the procedure.

Understanding the Geometry

Before preparing a symbol, study the symbol in terms of the dimensional parameters required to uniquely define the symbol, the ports or connect points required for the symbol, how the geometry of the symbol will be represented graphically, the origin of the symbol, the orientation of the symbol, and so forth.

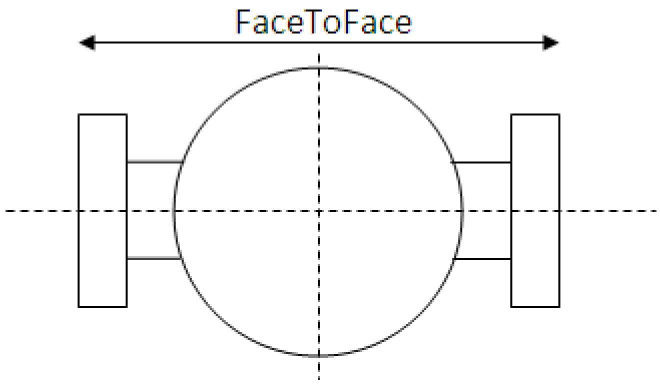
Determine the various aspects of the symbol to be drawn such as Physical, Insulation, Maintenance, and so forth. This means that you should decide whether to simply draw the symbol's physical representation only or whether to add the insulation graphics, maintenance space required, and so forth.

Referring to the example of a ball valve, Physical and Insulation aspects of the valve will be drawn.

Geometric Representation

The valve is drawn with a circular flange on the left hand side as a cylinder; next a cylinder, a sphere, a cylinder at the center, and another circular flange at the right hand side. This means three outputs are drawn to represent the Physical Aspect of the valve (Port1, ValveBody, and Port2). Please note that even though this example appears to represent a flanged ball valve, the code is generic enough to accommodate different end preparations such as welded, flanged,

threaded, and so forth. You can define a more complex geometric representation of the symbol as needed.



Dimensions

As a symmetrical valve, the face to face dimension is required to draw the valve. The dimensions required to draw the flange are obtained from the standard geometric data bulkloaded in the Smart 3D project's catalog database. This data is available in the AllCommon.xls for the various end preparations (Bolted, Male, or Female). The radius of the sphere is assumed to be a factor of the known dimensions. Hence the input parameters required those for representing the Physical Aspect (Face to Face dimension).

Orientation

The symbol is drawn with the origin (0, 0, 0) at the center of the valve. The left-hand side port is drawn along –X direction and the right-hand side port is drawn along +X direction.

Defining Ports on Symbols

Most symbols have at least one port, which is a point on a part that connects to a routed item, such as pipe or cableway. A port consists of an attachment point and direction, set of application properties, and physical geometry depiction. A different class of port is required for each type of routing item. For example, piping requires one type of port, while cableway requires another.

You define ports when you create a symbol and define the geometry of a part. In .NET, a function specifies the port type, name, attachment point, and attachment vector. The software places the ports based on the information in the geometry definition file for the part and the reference data for the part. The geometry definition file defines the port type, name, attachment point, and attachment vector. The reference data for the specific part (item of the part class) defines the remainder of the property values for the port.

Writing Code for the .NET Symbol

The Smart 3D software has the following geometry APIs, through which the outputs need to be generated.

- Line3D

- Circle3D
- Arc3D
- ComplexString3D
- Projection3D
- Revolution3D
- Torus3D
- BsplineCurve3D
- Cone3D
- Nozzle

The **SymbolGeometryHelper** contains primitive shapes such as **CreateCylinder**, **CreateCone**, **CreateCircularTorus**, **CreateSphere**, and so forth, which are used for creating different outputs.

You build the assembly to %OLE_SERVER%\Custom Symbols. You can build the assembly in any sub-folder under the Custom Symbols folder.

Deploying the Symbols

In the **Project Management** task, select the Catalog and then select **Tools > Update Custom Symbol Configuration**. This updates "CustomSymbolConfig.xml" with the ProgID of the new assembly and its location relative to the "%OLE_SERVER%" path.

Naming of the Symbol Definition

The definition name of a symbol should be unique in the database. It is recommended that the namespace of symbol definition class should be specified as follows:

<CompanyName>.SP3D.Content.<Specialization>.

For example, Ingr.SP3D.Content.Support

We also recommended that if a delivered symbol definition must be changed to meet a specific requirement, its namespace/symbol definition class name be changed so that the identity of the modified symbol is different (unique) from the one delivered in the software.

Bulkloading the Symbol

You should find an existing symbol similar in geometry to your custom symbol, nozzle location, and orientation, and so forth, and use its bulkload datasheet as a base in which to prepare a datasheet to bulkload the new symbol.

For example, to bulkload inline valves with two nozzles use the 'BALR' worksheet in the Piping.xls sheet. To bulkload an inline symbol when there is a change in diameter 'REDC', use the Piping.xls worksheet. You can have as many attributes as occurrence attributes (property values which can be changed at runtime), but this should be specified in the same row where the SymbolDefinition is mentioned.

A new symbol's non-graphic data should be added in a specified Microsoft Excel workbook. For example, Piping.xls should be edited for a new Piping symbol. Equipment.xls should be edited for a new Equipment symbol. Each of these Excel books defines the classes, parts, specifications, rules, and so forth. Some common workbooks, such as, AllCommon.xls and AllCodeLists.xls might need to be edited.

For each unique symbol, a separate worksheet should be added to the Microsoft Excel workbook with appropriate detail. You can use the Microsoft Excel workbook generated by the Part Definition Symbol Wizard. Provide nozzle information in the part class Excel sheet depending on the number of nozzles in the newly created symbol. Use 'A' to append, 'M' to modify, and 'D' to delete data into the Catalog database when bulkloading in Append mode.

Bulkloading the Piping Symbol

See the spreadsheets that are created for the ball valve delivered on your system as follows:

{Product Path}\CatalogData\BulkLoad\DotNetSampleDataFiles\Piping-DotNet.xls

{Product Path}\CatalogData\BulkLoad\DotNetSampleDataFiles\PipingSpecification-DotNet.xls

For more information not covered in this programming guide, including guidelines on bulkloading symbols, see *Loading Reference Data into the Catalog* in the *Reference Data Guide*.

Placing the Symbol

Complete the following steps for placing a piping symbol:

1. Open Smart 3D and navigate to the Piping task.
2. Click **Route Pipe** on the vertical toolbar.
3. Select the run starting point.

   If you select a feature located at the end of an existing run, the software continues the run of the selected feature. If you select an equipment nozzle, a point in space, or a point along a straight feature, the software prompts you to create a new pipe run.
4. On the **New Pipe Run** dialog box, type a name for the pipe run. If you do not type a name, the software automatically generates a name. Select the Piping System (Specification IC0031, IC0032, N0, N1) and NPD to be used for placing the pipeline. The NPD and Specification should be the same you used in the bulkload data for the symbol.
5. Click **OK** to close the **New Pipe Run** dialog box.
6. Select point to end routing of your pipe run.

7. Click **Insert Component**  on the vertical toolbar to insert a component.

   The Insert Component command adds valves, strainers, laterals, and other components to a pipe run. You can add components either during the routing of a pipe run or after the pipe has been routed.

   The system uses the pipe specification, nominal diameter of the selected pipe run, and the geometry of the insertion point to filter the available components. For example, if the insertion point is not at the end of a pipe run or at an equipment nozzle, turn components are not included in the list of available components. When you insert a component, the software generates any mating and connection parts required to connect the inserted part to the adjacent objects.

   When inserting components, you can use the **Tools > Pinpoint** and **Tools > Point Along** commands to position components precisely in a pipe run.

8. Select the component type and option in the **Type** and **Option** boxes.

9. Click to define the position of the component if you are placing it in a straight feature.

   If needed, change the position of the component using **Flip**, **Reference Position**, and **Angle** options.

10. Click **Finish**.

11. You can check the properties (input parameter values) by selecting the component and selecting **Edit > Properties**.

Creating an Advanced Symbol

Some symbols often use more advanced techniques in order to meet the different set of requirements posed on them. The following section provides details on these techniques.

Topics

Dynamic Outputs

Often the symbol outputs are not known before-hand and are determined dynamically during the computation of the symbol. For example, concerning a symbol to create the geometry of a stair, the number of outputs is dependent on the span of the stair and pitch.

The 3D API framework allows addition of such outputs dynamically. Adding dynamic outputs to the symbol by:

1. Informing symbol machinery that you'll be creating variable outputs.

   If you derive your symbol definition class from a business object specific base class (that is, StairSymbolDefinition), this step is not needed and you can jump directly to step 2.

   Add *VariableOutputs* attribute on your class.

   ```
   <VariableOutputs()>
   ```

2. Constructing the output object; a symbol output must be persistent; that is, it must be created with a valid database connection.

3. An output object is added to the symbol outputs with a unique name.

   ```
   'Add an output object myOutputObject named "MyOutputName"
   'to the aspect m_oSimplePhysicalAspect.
      m_oSimplePhysicalAspect.Outputs.Add("MyOutputName ",
   myOutputObject)
   ```

Custom Weight and Center of Gravity (COG)

If a symbol is responsible for computing weight and center of gravity (COG), the symbol should compute the volume and COG of the geometric outputs in the ConstructOutputs module. This results in better performance rather than getting the outputs later and performing calculation.

Catalog parts can have single or multiple materials. For example, a pipe part has single material, but a ladder can have a variety of materials for the frame, cage, safety gate, support legs, bolts, and so forth. Weight and COG evaluation for the parts need to consider all the individual materials.

The 3D API framework supports custom weight COG calculation for both kinds of parts. Evaluating weight COG in the symbol consists of:

1. Get the net volume and COG for the geometric outputs of the symbol for each material.

2. Construct a VolumeCOG named output object for each material and add it as output to the symbol.

   ```
   'Create a VolumeCOG object for a HandRail symbol.
   oHandRailVolCOG = New VolumeCG(oConnection, dTotalVolume, dCOGX,
   dCOGY, dCOGZ)
   m_oSimplePhysicalAspect.Outputs["VolumeCOG"] = oHandRailVolCOG
   ```

3. Symbols which handle parts with single material only need to complete previous steps 1 and 2. Weight COG will be calculated by the business object from this named output.

4. Symbols which handle parts with multiple materials need to realize ICustomWeightCG. This interface provides methods to evaluate weight COG or the part by the symbol. In this case, the symbol gets the outputs from the output collection and the materials from the part to calculate the net weight COG.

```
EvaluateWeightCG(ByVal oBO As BusinessObject) Implements
ICustomWeightCG.EvaluateWeightCG
      'Get VolumeCOG output.
      'Get the output from the symbol output collection using the
      'helper method provided on SymbolHelper.
       oObject = SymbolHelper.GetSymbolOutput(oBO, "SimplePhysical", _
            "VolumeCOG")
             If Not oObject Is Nothing Then
             oHandRailVolCOG = DirectCast(oObject, VolumeCG)
             dVolume = oHandRailVolCOG.Volume
             dCOGX = oHandRailVolCOG.COGX
             dCOGY = oHandRailVolCOG.COGY
             dCOGZ = oHandRailVolCOG.COGZ

             'Evaluate weight from output volume and the material
             'properties on the given user interface.
             dWeight = EvaluateWeightFromVolume(oBO, dVolume, _
                   sIJUAHandRailTypeAProps)

             'Set the net weight and COG on the Business Object using
             'helper method provided on StructureSymbolDefinition.
             SymbolHelper.SetWeightAndCOG(oBO, dWeight, dCOGX, dCOGY,
dCOGZ)

             End If
```

Custom Evaluation of Origin and Orientation

Often parts need to be positioned and oriented correctly based on the given inputs. For example, a stair needs to be positioned and oriented based on the TopSupport, SideReference, and BottomSupport inputs selected by you.

The 3D API framework provides an interface ICustomEvaluate which should be realized by the symbol to support evaluation of origin and orientation for parts.

Evaluating origin and orientation of a part involves the following steps:

1. Realize ICustomEvaluate on the symbol.
2. Construct the transformation matrix based on the given inputs.

The following code example demonstrates how to construct the transformation matrix from the vectors based on the geometric inputs and the position value given for a stair:

```
EvaluateGeometry(ByVal oBO As BusinessObject, bPartChanged As bool,
bGeomInputChanged As bool, bPropertyValueChanged As bool)

…
'Initialize the matrix to identity.
oMatrix = New Matrix4X4()
'Construct a double array and set the actual double values for the
local
'x, y, z vectors in the transformation matrix.
'Also set the translation component.
Dim dArrMatrix As Double() = New Double(15) {}
…
```

```
'Set the double array on the matrix.
oMatrix.Set(dArrMatrix)
3.    Set the transformation matrix on the Business Object.
The following code example shows how to set the transformation matrix
on the Ladder object.

'Set the orientation matrix on the ladder or stair.
Dim ladderStairObject As StairLadderBase = DirectCast(oBO,
StairLadderBase)
ladderStairObject.Matrix = oMatrixOrientation
```

Custom Foul Check

Some business objects delegate foul check to the symbol which allows the symbol writer to override the default implementation. For example, a footing assembly symbol may need to return all the supported member parts as connected parts to suppress interference between them and the footing. A footing component symbol might need to return non-participant for interference check to avoid duplicate interference reporting by the components.

The 3D API framework provides an interface **ICustomFoulCheck** which should be realized by the symbol to support custom behavior of foul check for parts.

Supporting custom behavior for foul check on a symbol involves the following steps:

1. Realize **ICustomFoulCheck** on the symbol.
2. **GetConnectedParts** should return the collection of connected parts or return null.
3. **GetInterferenceType** should return the interference type.

The following code example shows how to implement **GetConnectedParts** and **GetInterferenceType** for the footing assembly symbol:

```
GetConnectedParts(ByVal oBO As BusinessObject) As ReadOnlyCollection(Of
BusinessObject)
…
    'Get all the supported objects from the footing.
    Dim oConnectedPartsList As New List(Of BusinessObject)()
…
    'Get all the supported objects from the footing.
    'For each supported object, if it is a MemberSystem, get its parts
and add them to the list.
…

    Return New ReadOnlyCollection(Of
BusinessObject)(oConnectedPartsList)
End Function

GetFoulInterfaceType(ByVal oBO As BusinessObject) As FoulInterfaceType
    'Footing assembly is participant in interference.
    Return FoulInterfaceType.Participant
End Function
```

Custom Mirror

Some business objects delegate the mirror implementation for the symbol code which allows the symbol writer to override the mirror behavior of a specific part; for example, a stair might need to be flipped around the top support on mirror.

The 3D API framework provides an interface **ICustomMirror** which should be realized by the symbol to support custom mirror behavior for mirroring parts.

Supporting custom behavior for mirror on a symbol involves the following steps:

1. Realize **ICustomMirror** on the symbol.
2. Set properties which effect mirror behavior inside the **Mirror** method.

The following code example shows how to implement mirror for a ladder:

```
Mirror(ByVal oBusinessObject As BusinessObject, ByVal
oBusinessObjectOrig As BusinessObject, ByVal oMirrorPlane As IPlane,
ByVal oTransformMatrix As Matrix4X4, ByVal bIsCopy As Boolean)
…

    'Add custom behavior for mirror here based on mirror behavior.

    Dim iMirrorBevahior As Integer =
CInt(SymbolHelper.GetLongProperty(DirectCast(oPart, BusinessObject),
SPSSymbolConstants.IPART, SPSSymbolConstants.MIRRORBEHAVIOROPTION))
    If iMirrorBevahior = SPSSymbolConstants.REPLACEMENTPARTVALUE Then
…
```

Custom Property Management

Client tier code for some business objects can use the symbol code to verify the validity of values given for properties on placement and during edit through the property pages. For example, a stair can only support angle values within a certain range. Also, a ladder symbol which only supports a vertical ladder might need to have the angle field read-only *only* in the client tier.

The 3D API framework provides an interface **ICustomPropertyManagement** which should be realized by the symbol to support validation and management of the properties on the part.

Implementation of custom property management involves the following steps:

1. Realize ICustomPropertyManagement on the symbol.
2. OnPreLoad is called immediately before the properties are loaded in the property page control. Any change to the display status of properties can be done here.

   Following code example demonstrates how to set the display status of a property to read-only:

   ```
   OnPreLoad(ByVal oBusinessObject As BusinessObject, ByVal
   CollAllDisplayedValues As ReadOnlyCollection(Of PropertyDescriptor))
       …
       'Validate each property value.

       For i As Integer = 0 To CollAllDisplayedValues.Count - 1
           Dim oPropDescr As PropertyDescriptor =
   CollAllDisplayedValues(i)
   ```

```
            Dim oPropValue As PropertyValue = oPropDescr.[Property]
            Dim sPropName As String = oPropValue.PropertyInfo.Name

            'Make all these properties read-only.
            Select Case sPropName
                Case "MyReadOnlyPropertyName"
                    oPropDescr.[ReadOnly] = True
                    Exit Select
            End Select

            …

        Next
    …
```

3. **OnPropertyChange** is called each time a property is modified. Any custom validation can be done here.

Following code example shows how to validate the value of a property on change in the property value.

```
OnPropertyChange(ByVal oBusinessObject As BusinessObject, ByVal
CollAllDisplayedValues As ReadOnlyCollection(Of PropertyDescriptor),
ByVal oPropToChange As PropertyDescriptor, ByVal oNewPropValue As
PropertyValue, ByRef sErrorMessage As String) As Boolean
…

    sInterfaceName =
oPropToChange.[Property].PropertyInfo.InterfaceInfo.Name
    sPropertyName = oPropToChange.[Property].PropertyInfo.Name
…
    'Check the property value.
        If sErrorMessage.Length > 0 Then
            bOnPreLoad = False
            Exit For
        End If
…
```

To Do Record Messages

A content writer developing a symbol must write the code to create the output objects. In some cases, the given set of input values might be invalid or semi-valid. In such a case, the content writer can create a To Do Record (TDR) associated with the symbol occurrence. The TDR causes the symbol to be added to the To Do List with a message. An error message would be displayed for an invalid set of inputs. A warning message would be displayed for a semi-valid set of inputs.

Prior to Version 2011 R1 (9.1), a TDR for a symbol requires the message to be defined in a codelist table. The codelist table name and codelist index must be passed into the function by raising an exception that creates the TDR. The existing exception classes are: SymbolErrorException and SymbolWarningException. The exception contains the data that is needed to create a TDR. This method of creating a TDR is still supported, but is deprecated and should no longer be used when writing new .NET symbols.

Posting errors from .NET symbols using exceptions has an important drawback in that after the exception is raised, the remaining code within the .NET symbol is not executed. This does not work well for warnings which need to continue execution after the warning is posted.

Beginning in Version 2011 R1 (9.1), a TDR with a string message can be created by setting the new ToDoListMessage property on the base class of the symbol. The new messages are uniquely identified by the combination of the message module name and the message number. A message module is a logical collection of messages. The names of modules must be unique across the entire product. Every message is identified with a message number. Message numbers must be unique within a module.

The recommended conventions are:

- Each component defines one or more message modules. Each message module corresponds to a resource file containing localized messages.
- Customers who add error messages for their custom content should use message module names that identify the company name (for example, AcmeErrorMsgs).
- Use the resource ID as the message number.

A new property has been added to the CustomSymbolDefinition base class. To create a TDR, the symbol code must set this new property and should not raise an exception.

```
public abstract class CustomSymbolDefinition
{
    public ToDoListMessage ToDoListMessage
    {
        get { return m_oTDLMessage; }
        set { m_ oTDLMessage = value; }
    }
```

The data type of the new property is a class as shown below.

```
public class ToDoListMessage
{
    // Constructors
    // Construct a very simple ToDoListMessage specifying only type and message text.
    //    This constructor is intended for custom content writers that do not localize
    //    their messages. A default module name and message number are supplied by this
    //    constructor.
    //    ArgumentException is raised if an empty string is passed for text.
    public ToDoListMessage(ToDoMessageTypes type, string text)
    // Construct a simple ToDoListMessage when the objectToUpdate is not needed.
    //    This constructor is intended for most application developers and content writers
    //    who localize their messages and provide a help topic.
    //    ArgumentException is raised if an empty string is passed for moduleName or text.
    public ToDoListMessage(ToDoMessageTypes type, string moduleName, int number, string text)
    // Construct a ToDoListMessage including an objectToUpdate.
    //    This constructor is intended for the rare case where an object to update that is
    //    different from the symbol must be specified.
    //    ArgumentException is raised if an empty string is passed for moduleName or text.
    //    ArgumentNullException is raised if objectToUpdate is Null.
    public ToDoListMessage(ToDoMessageTypes type, string moduleName, int number, string text,
BusinessObject objectToUpdate)
    // Properties
    // The Type property identifies the message as an error or a warning.
    public property ToDoMessageTypes Type { get; }
    // The ModuleName property identifies a logical grouping of messages.
    //    Each component/customer defines their own module name.
    //    Module names must be unique across the product.
    public property string ModuleName { get; }
    // The Number property identifies a specific message within a module.
    //    The combination of module name and message number uniquely identifies the message.
    public property int Number { get; }
    // The Text property holds the localized message text including any contextual data.
    public property string Text { get; }
    // The ObjectToUpdate property holds a reference to the object to be updated when
    //    the user clicks on the Update command in the To Do List dialog.
    //    If the object to update is not specified, the symbol occurrence itself is updated.
    public property BusinessObject ObjectToUpdate { get; }
}

// Describes the possible types of to do messages.
public enum ToDoMessageTypes
{
    // The symbol failed to compute the output objects.
    ToDoMessageError = 1,
```

```
    // The symbol encountered problems while computing the output objects.
    ToDoMessageWarning = 4,
}
```

Sample Usage

To create an error To Do Record:

```
string strMessageModule = "EquipProcessMsgs";
int nMessageNumber = 5;
string strMessageText = "Unable to construct tank geometry due to conflicting parameter values";
oSymbol.ToDoListMessage = new ToDoListMessage(ToDoMessageTypes.ToDoMessageError, strMessageModule, nMessageNumber,
strMessageText);
```

To create a warning To Do Record:

```
string strMessageModule = "StructStairMsgs";
int nMessageNumber = 43;
string strMessageText = "Warning: stair pitch is too steep";
oSymbol.ToDoListMessage = new ToDoListMessage(ToDoMessageTypes. ToDoMessageWarning, strMessageModule, nMessageNumber,
strMessageText);
```

Checking the Status of Nested Symbols

There are 3D APIs to place and update a nested symbol from an outer symbol. You must check the status of the nested symbol after an update to see if the update resulted in success, failure, or warning.

A read-only property called ToDoListMessage is available in SymbolOccurrence. After calling update() on the nested symbol, this property needs to be checked to know the status of the update. When the symbol updates successfully, this property is null. When the Update() results in a warning or error, the ToDoListMessage property is not null. The following example shows how to check the status of Update().

```
Dim oBox As SymbolOccurrence
    oBox = New SymbolOccurrence(oConnection,
"SP3DBallValve,Ingr.SP3D.Piping.NetBox", "", True)

    oBox.SetInputDouble("Xmax", 0.6)
    oBox.SetInputDouble("Ymax", 0.6)
    oBox.SetInputDouble("Zmax", 1.1)

    oBox.Update()
    'Check the status of update
    Dim oBoxTDLMsg As ToDoListMessage
    oBoxTDLMsg = oBox.ToDoListMessage
    If Not oBoxTDLMsg Is Nothing Then
       'Create a ToDoList Message on outer symbol
       ToDoListMessage = New ToDoListMessage(oBoxTDLMsg.Type,
oBoxTDLMsg.Text)
    End If
```

If the outer symbol does not check the status of Update() for the nested symbol, and therefore does not create a ToDoListMessage on itself, no To Do record is created. No To Do record is created on the inner symbol because the Update() method on the inner symbol does not create a To Do record.

NOTE Any symbol occurrence (outer or inner) created using 3DAPI has the property 'ToDoListMessage' which can be used in 3DAPI.

If there are any existing .NET symbols that use nested symbols, they need to be modified to check the status of nested symbols after Update(). In case there is an error, appropriate action

(such as creating a ToDoListMessage on outer symbol as shown in the example above) needs to be taken.

Creating .NET Symbols using the Symbol Wizard

Deployment

To help you create new symbols, the Symbol Wizard is delivered with the software. The wizard is an executable, runs as a stand-alone application, and is delivered as follows:

```
{Product Path}\Core\Container\Bin\Assemblies\Release\SymbolWizard.exe
```

For more information see its integrated context sensitive (F1) help.

The wizard implementation is language neutral. It uses style sheet templates and XML transformation to generate a .NET symbol definition for any programming language (VB.NET, C#.) needed by the symbol author. The style sheet templates are delivered as follows:

```
{Product Path}\CommonApp\SOM\Client\Services\SymbolWizard\Templates
```

One VB.NET style sheet for each Solution, Project, AssemblyInfo, and the Symbol class is delivered, but these can be replaced with other programming language templates.

Workflow

The workflow for using the wizard to create a new symbol consists of the following steps:

1. Identify the .NET symbol project and location.
2. Specify either a new or existing project in which to add the .NET symbol.
3. Provide a Namespace and the symbol class name. Define inputs to the symbol in the inputs grid. See the section on Naming of the Symbol Definition for naming guidelines.
4. Select aspects defined by the symbol.
5. Define outputs for each aspect.

On finish, the new .NET Symbol class is created in the target project.

Useful Tips for Symbol Definition Coding

Application-specific Symbol Definition Base Classes

By default, the symbol class created by the wizard inherits from **CustomSymbolDefinition**. Smart 3D provides application and business object-specific symbol definition base classes which provide some useful functionality; for example, **StructureSymbolDefinition**, **LadderSymbolDefinition**, and so forth. A symbol should inherit from one of these base classes to help implement all the necessary behavior on the symbol.

SymbolGeometryHelper

As previously mentioned, the **SymbolGeometryHelper** provides a useful API to create geometry primitives for symbol output. Some of these functions include: **CreateCylinder()**, **CreateCone()**, **CreateCircularTorus()**, and so forth.

Migrating an Existing Symbol to .NET

You should consider the following questions when making decisions for migrating existing COM content to .NET:

1. Do I have access to VB6, VC++, or Visual Studio in which to develop?
2. Does the symbol need to run in a 64 bit version of the application yet?
3. Are there features of .NET or the new 3D API in which I need to take advantage?

If some or all of the above are requirements, then you should consider migrating the symbol to .NET.

This following section provides guidelines on how to migrate an existing symbol to .NET.

Migration Wizard

The **Symbol Wizard** allows symbol authors to create new symbols or migrate selected symbol definitions to .NET.

Workflow

Workflow for using the Symbol Wizard for migrating existing Symbols is:

1. Identify the .NET Symbol Project and location.
    a. Specify either a new project in which for the .NET Symbol to be created or add the Symbol to an existing project.
2. Provide a Namespace and the new Symbol class name. See the section on Naming of the Symbol Definition for naming guidelines.
3. Identify the existing symbol(s) to be migrated
    a. Select a .dll which contains existing symbols.
    b. Select one or more required existing symbols from the list of available symbols.

On finish, the new .NET Symbol class will be added to the new or existing project.

Migrated .NET Symbol Class

The migrated .NET Symbol class inherits the following information from the old symbol:

- Inputs
- Aspects
- Outputs
- A new Symbol Definition format of the ConstructOutputs() method stub will be provided where the logic needs to be added to create the necessary outputs for each aspect.

Creating a Custom Assembly

Custom Assembly represents an extension of a 3D symbol where more than geometry can be produced as outputs. You can include other first class business objects, such as nozzles on equipment or structural members in an equipment foundation. The Custom Assembly inherits from the base symbol definition class so the same deployment and advanced extensions exist for the Custom Assembly.

Defining a Custom Assembly

Defining a Custom Assembly implies inheriting from an application provided by the Smart 3D Custom Assembly base class such as **EquipmentAssemblyDefinition**, **FootingCustomAssemblyDefinition**, or **EquipmentFoundationCustomAssemblyDefinition**. These base classes provide a basis for working with the business object and existing Smart 3D user interface.

Defining Assembly Outputs

Assembly outputs are declared as fields of your Custom Assembly:

```
' Declare assembly outputs
<AssemblyOutput(1, "Pier")> _
Public m_Pier As AssemblyOutput
<AssemblyOutput(2, "Grout")> _
Public m_Grout As AssemblyOutput
```

The field variable must be declared `Public` and have a defining attribute providing it with a unique index and name. Omitting the defining attribute from the output simply ignores the declared output.

Creating / Evaluating Assembly Outputs

Construction and modification of assembly outputs occurs in the **EvaluateAssembly** method. This method is invoked immediately following the symbol's **ConstructOutputs** method. All parameter inputs and symbol outputs are available for you to access when manipulating the assembly outputs.

Creating Assembly Output

You create assembly outputs by setting the declared field assembly output's field variable **Output** property to a persistent business object. **EvaluateAssembly** is invoked when the outputs are to be created and anytime the assembly needs to be evaluated. Therefore, it is the responsibility of the developer to determine whether the output is already generated. A typical pattern for creating an assembly output might appear as:

```
If (m_NozzleSuction.Output Is Nothing) Then
      m_NozzleSuction.Output = ConstructNozzleSuction(oSP3DConnection)
End If
```

The code *always* checks whether the output already exists and only constructs the output when it is `Nothing`. Failure to make this check results in an exception indicating that the output already exists.

Modify Assembly Outputs

As noted earlier, output will already exist with subsequent invocations of the **EvaluateAssembly** method. To modify the output, cast the output to the constructed business object class and manipulate the object:

```
<AssemblyOutput(2, SPSSymbolConstants.Pier)> _
Public m_oPierAssemblyOutput As AssemblyOutput
.
.

Public Overrides Sub EvaluateAssembly()
Dim oPierComponent As FoundationComponent = Nothing

' construct the pier (if not generated yet)
If m_oPierAssemblyOutput.Output Is Nothing Then
     oPierComponent = CreateComponent(SPSSymbolConstants.Pier)
     m_oPierAssemblyOutput.Output = oPierComponent
Else
     oPierComponent = DirectCast(m_oPierAssemblyOutput.Output,
FoundationComponent)
End If

oPierComponent.Origin = New Position(1, 2, 3)
.
.
```

Optional Assembly Outputs

Even though an assembly output has been declared, it does not imply that you must create an output. Not constructing an output indicates the output is not required. Additionally, if an output already exists you can remove the output:

```
If m_oPierAssemblyOutput.Output Is Nothing Then
      m_oPierAssemblyOutput.Delete()
End If
```

Now using this code, the output no longer exists. When the evaluation method is invoked again, you can decide whether to construct the output again.

Accessing Object Inputs

While constructing and evaluating the assembly outputs, access to the object inputs may be required; such as the structural member where the footing is to be placed.   These inputs must be retrieved from the business object using the **Occurrence** property on the Custom Assembly:

```
'accessing the footing business object
Dim oFooting As Footing = DirectCast(Occurrence, Footing)
```

Direct casting of the occurrence to the specific business object provides access to the object-specific properties which would include the inputs (supported members and supporting members in the example above).

Allowing End-Users to Delete Assembly Outputs

By default an end-user will not be able to remove the assembly outputs of a Custom Assembly without removing the parent business object. To allow independent removal of an output, you must set the property **CanDeleteIndependently** to `True`. By setting this property to `True`, within your **EvaluateAssembly** method, an end-user will be able to delete your assembly output independently of the parent business object. By allowing this behavior, the object construction is slightly complicated because a check now will be necessary to determine whether the user has deleted your output. The object construction checks would now appear as:

```
' Construct the pier (if not generated yet).
Dim oPierComponent As FoundationComponent = Nothing
If m_oPierAssemblyOutput.Output Is Nothing Then
    If Not m_oPierAssemblyOutput.HasBeenDeletedByUser Then
         m_oPierAssemblyOutput.CanDeleteIndependently = True
         oPierComponent = CreateComponent("Pier")
         m_oPierAssemblyOutput.Output = oPierComponent
    End If
Else
    oPierComponent = DirectCast(m_oPierAssemblyOutput.Output,
FoundationComponent)
End If
```

Notice the additional check as to whether the output was deleted by the user with the **HasBeenDeletedByUser** property. This property will be true when the output existed at one time and was explicitly removed by the user. Also, notice the line of code that set the **CanDeleteIndependently** property to `True`, which allowed the end-user to delete the assembly output in the first place.

Dynamic Outputs

Similar to a symbol, Custom Assembly provides for dynamic outputs (that is, outputs whose count may vary dynamically at runtime). An example of this might be the structural member legs of an equipment foundation. Declaring a dynamic output can appear similar to the following:

```
'Declaring a dynamic structural member leg output.
<AssemblyOutput(1, "FoundationMemberlegs”)> _
Public m_objFoundationLegs As AssemblyOutputs
```

The definition of a field variable of type AssemblyOutputs with same AssemblyOutput defining attribute as present for singularly declared assembly outputs. AssemblyOutputs inherits from List<BusinessObject>; hence, AssemblyOutptus is a collection (i.e., list) of BusinessObjects. The code within the EvaluateAssembly method would add and subtract business objects from this list based on the needed count:

```
<InputString(2, "FoundationShape", "Foundation Shape", "Rectangle")> _
Public m_sFoundationShape As InputString

<AssemblyOutput(1, "FoundationLegs")> _
Public m_objFoundationLegs As AssemblyOutputs

'Evaluate assembly outputs.
Public Overrides Sub EvaluateAssembly()
    Dim iLegCount As Integer
    iLegCount = 0

    If m_sFoundationShape.Value = "Rectangle" Then
         iLegCount = 4
    ElseIf m_sFoundationShape.Value = "Hexagon" Then
         iLegCount = 6
    End If

    ' Current leg count.
    Dim iCurrentLegCount As Integer
    iCurrentLegCount = m_objFoundationLegs.Count

    Dim iCnt As Integer
    If iCurrentLegCount = iLegCount Then ' No change.
    ElseIf iCurrentLegCount < iLegCount Then ' Need to add some legs.
        For iCnt = iCurrentLegCount + 1 To iLegCount
            m_objFoundationLegs.Add(New Member(...
        Next
    Else ' remove some legs
        For iCnt = iCurrentLegCount To iLegCount + 1 Step -1
            m_objFoundationLegs.RemoveAt(iCnt)
        Next
    End If
.
.
```

Bulkloading a Custom Assembly

Bulkloading a Custom Assembly varies slightly from bulkloading a symbol definition:

Head	Name	SymbolDefinition	Definition	Sy
Start				
	CircularPierAndRectangularlabFooting		FootingCustomAssemblies,Ingr.SP3D.Content.Structure.PierAndSlabFtgDef	Sn
End				

The Custom Assembly ProgID (in this case, assembly .dll name with custom assembly class name and its namespace), defined in the **Definition** field of a bulkloaded spreadsheet, indicates a Custom Assembly.

When this field is present in the spreadsheet for items such as equipment, footings, equipment foundations, etc., it is expected that the **Definition** field should be completed with the name of a Custom Assembly. The **SymbolDefinition** is left blank (that is, this field is ignored).

Creating and Scheduling Custom Batch Jobs

To use custom batch jobs in Smart 3D, you must follow these steps:

1. Create a custom batch job that you can schedule, and run it through the Smart 3D batch framework.
2. Register the custom batch job so that the administrator can configure queues for the job.
3. Submit the custom batch job through a custom command.

Creating a Custom Batch Job

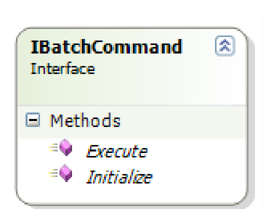
Create a custom batch job by creating a new class in the job that implements **IBatchCommand**.

IBatchCommand Interface:

In the new class that implements **IBatchCommand** interface, define two methods:

- **Initialize** - Initializes all the arguments that are submitted to Intergraph Batch Server. It also initializes all the variables using the .xml file that is generated after submitting the job to IBS.
- **Execute** - Executes the custom batch job process. You must implement the custom batch job in this method.

NOTE For more information on defining the **Initialize** and **Execute** methods, see the Visual Studio integrated reference documentation on **BatchCommand.cs.**



Configuring the Queues for Custom Batch Jobs

To configure a queue for a custom batch job, you must retrieve the command ProgID, job name, and job ProgID and copy them into the CustomBatchJobsDetails.xml file, located in [Product Directory]\SharedContent\Xml.

Below is the sample format of the CustomBatchJobsDetails.xml. If this file does not exist, then create a new file in the below format.

```
<JOB>
      <JobType>Check Database Integrity</JobType>
<JobDescription>To check the S3D database integrity and
consistency</JobDescription>
<JobProgId>BatchSvcCmds,Ingr.SP3D.ProjectMgmt.Client.Commands.BatchR3DM
ultipleUpdate </JobProgId>
      <IsModelSpecific>True</IsModelSpecific>
</JOB>
```

- **JobType** - Defines the name of the job. The name should be unique.
- **JobDescription** - Defines the job description.
- **JobProgId** - Defines the ProgID created for the custom batch job.
- **IsModelSpecific** - Defines the type of job. Set **IsModelSpecific** to **True** to specify the job as a model-specific job, and set **IsModelSpecific** to **False** to specify the job as a site-specific job.

  NOTE Custom batch jobs can only be created for models, so set **IsModelSpecific** to **True**.

Register this custom batch job so that the administrator can configure queues for the job in Project Management.

Scheduling a Custom Batch Job

You can schedule the custom batch job using the **Schedule Data Consistency check** dialog box.

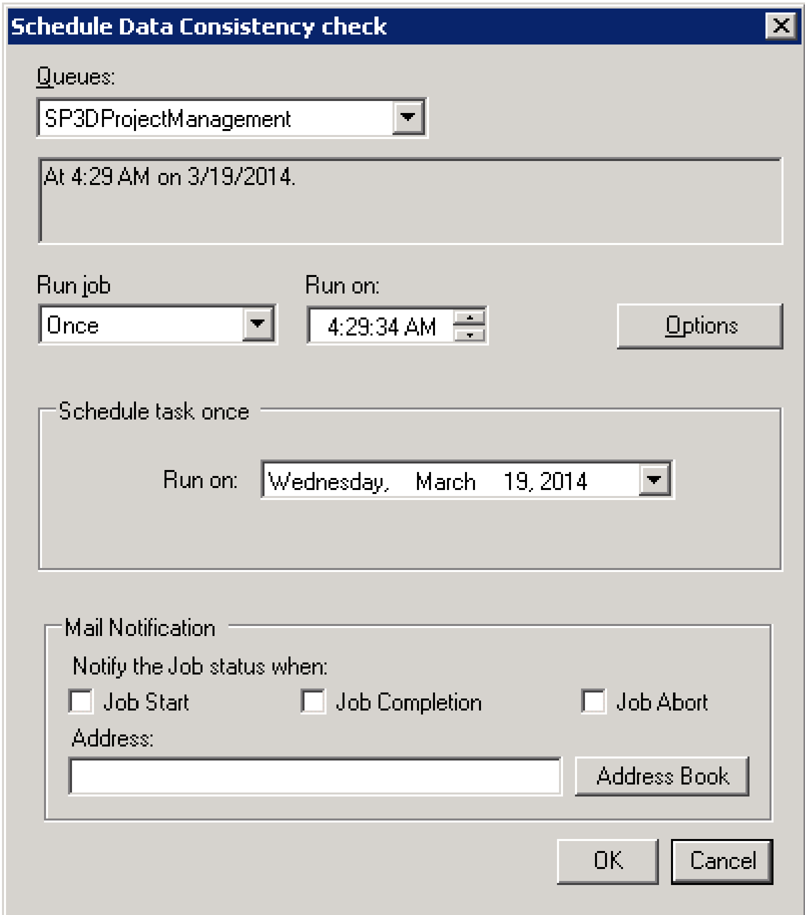
When you submit a job for any custom batch commands in .NET, you must first define a **JobCreator** class, which allows the API to display the **Schedule Data Consistency check** dialog box. The **JobCreator** class also allows you to submit the batch job to the Intergraph Batch Server.

The **Schedule Data Consistency check** dialog box provides information to the Intergraph Batch Server, such as the queue on which the batch job is submitted, scheduling properties, and mail notification options.

1. Create a **JobCreator** class to use the following APIs to submit a custom batch job:
   - **Initialize** - Initializes the job.
   - **ShowSchedule** - Opens the *Schedule Data Consistency Check Dialog Box* (on page 35).

   *After the above APIs are defined, the **Schedule Data Consistency Check** dialog box*

displays:



2. After selecting all the inputs, click **OK**.

   *The **Ok_Clicked** event handler displays, allowing you to create the required inputs to execute the batch job in .xml format.*

3. In the **OK_Clicked** handler, call the **AddBatchRequest** method, which takes command line arguments in .xml format.

4. Create an .xml file with all the required inputs for the job and save your changes.

Schedule Data Consistency Check Dialog Box

Queues

Specifies the queue on which the job is run.

Run job

Specifies how often the job is run.

Run on (time)

Specifies the time to run the job.

Options

Defines further options for scheduling the job.

Run on (date)

Specifies the date on which to run the job. Use this option if you want the job to be run once.

Notify the Job Status when

Select **Job Start** to receive the job status at the job start, **Job Completion** to receive the job status once it is complete, or **Job Abort** to receive the job status when it is aborted.

Address

Specifies the email addresses to which the **Job Status** notifications are sent.

SECTION 3

Creating Symbols in Solid Edge

Using Solid Edge, you can do the following:

- Model different types of equipment representations.
- Use the equipment representations for different types of interference checking.
- Combine multiple representations for a single piece of equipment component into an assembly file.

NOTE Solid Edge symbols are only supported by the software when they are used to create equipment. Solid Edge symbols are not supported by Smart 3D for other tasks.

The **Occurrence Name** suffix of each part identifies the representation or aspect for the part. You can find the available aspects listed in the **Aspect Code** sheet of the AllCodeLists.xls workbook. The following table lists the common Solid Edge file names, associated aspects, and example occurrence names. The assembly (.asm) files store the aspects. An assembly file contains all the representations. You list the .asm file name on the part class sheet in the Excel workbook. The **Occurrence Name** of a .par file defines the aspects in an assembly file.

Codelist Value	Representation	File Name Example	Occurrence Name Example
0	Simple physical	MySymbol1.par	MySymbol1_0.par:1
0	Simple physical	MySymbol2.par	MySymbol2_0.par:1
4	Detailed physical	MySymbol3.par	MySymbol3_4.par:1
6	Operation	MySymbol4.par	MySymbol4_6.par:1
7	Maintenance	MySymbol5.par	MySymbol5_7.par:1
8	Reference Geometry	MySymbol6.par	MySymbol6_8.par:1

NOTE The suffix in the **Occurrence Name** represents the aspect. The software assumes that the aspect is **Simple physical** if you do not specify an aspect or if you specify any characters other than the codelist values in the suffix of the **Occurrence Name**.

To control the sizes of the parts, define dimension variables and user-defined variables in Solid Edge. Map the user-defined variables to Smart 3D properties in the Equipment.xls workbook. Each part class sheet in the workbook must contain a column for each user-defined variable in Solid Edge.

For a dimension to be a driving variable, define it as a user-defined variable in Solid Edge, and then define that variable as an occurrence property using the syntax **oa:AttributeName** in the Equipment.xls workbook.

You cannot move nozzles on parts within Smart 3D. When you model the parts, you can use a macro to define nozzles, or ports, in Solid Edge. The macro assigns a type and a name to each

port. For more information, see *Create Solid Edge parts and assemblies for use in Smart 3D* (on page 37).

See Also

Create Solid Edge parts and assemblies for use in Smart 3D (on page 37)
Create Smart 3D reference data for use with Solid Edge components (on page 39)
Load and revise Smart 3D reference data (on page 42)
Place and modify Solid Edge components in Smart 3D (on page 43)

Create Solid Edge parts and assemblies for use in Smart 3D

Smart 3D places Solid Edge assemblies. These files use a .asm extension. You must store these files on a shared network drive so that multiple users can access them through Smart 3D. Typically, the file location is:

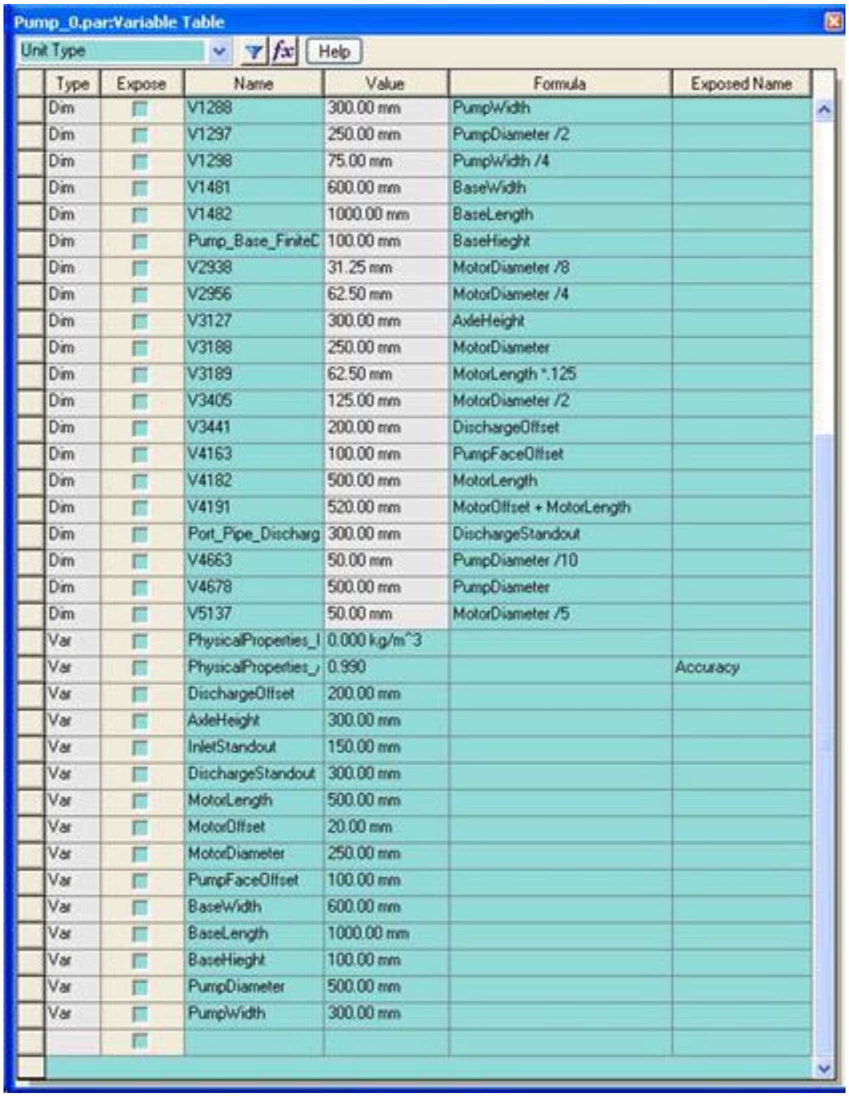
```
\\[Server Name]\Symbols\SolidEdgeParts
```

1. Name the part files in the Solid Edge assemblies used in Smart 3D using the following conventions:

Example File Name	Smart 3D Display Aspect Representation
MySymbol_0.par	Simple Physical
MySymbol_4.par	Detailed Physical
MySymbol_5.par	Insulation
MySymbol_6.par	Operation
MySymbol_7.par	Maintenance

This example uses the assembly file name MySymbol.asm.

2. Create a named variable in the Solid Edge variable table of the part to define any Smart 3D-driven dimension.



Pump_0.par:Variable Table

Unit Type | Help

Type	Expose	Name	Value	Formula	Exposed Name
Dim		V1288	300.00 mm	PumpWidth	
Dim		V1297	250.00 mm	PumpDiameter /2	
Dim		V1298	75.00 mm	PumpWidth /4	
Dim		V1481	600.00 mm	BaseWidth	
Dim		V1482	1000.00 mm	BaseLength	
Dim		Pump_Base_FiniteE	100.00 mm	BaseHieght	
Dim		V2938	31.25 mm	MotorDiameter /8	
Dim		V2956	62.50 mm	MotorDiameter /4	
Dim		V3127	300.00 mm	AxleHeight	
Dim		V3188	250.00 mm	MotorDiameter	
Dim		V3189	62.50 mm	MotorLength *.125	
Dim		V3405	125.00 mm	MotorDiameter /2	
Dim		V3441	200.00 mm	DischargeOffset	
Dim		V4163	100.00 mm	PumpFaceOffset	
Dim		V4182	500.00 mm	MotorLength	
Dim		V4191	520.00 mm	MotorOffset + MotorLength	
Dim		Port_Pipe_Discharg	300.00 mm	DischargeStandout	
Dim		V4663	50.00 mm	PumpDiameter /10	
Dim		V4678	500.00 mm	PumpDiameter	
Dim		V5137	50.00 mm	MotorDiameter /5	
Var		PhysicalProperties_I	0.000 kg/m^3		
Var		PhysicalProperties_/	0.990		Accuracy
Var		DischargeOffset	200.00 mm		
Var		AxleHeight	300.00 mm		
Var		InletStandout	150.00 mm		
Var		DischargeStandout	300.00 mm		
Var		MotorLength	500.00 mm		
Var		MotorOffset	20.00 mm		
Var		MotorDiameter	250.00 mm		
Var		PumpFaceOffset	100.00 mm		
Var		BaseWidth	600.00 mm		
Var		BaseLength	1000.00 mm		
Var		BaseHieght	100.00 mm		
Var		PumpDiameter	500.00 mm		
Var		PumpWidth	300.00 mm		

3. If necessary, add ports (nozzles) to the Solid Edge parts. Use the following convention for Smart 3D to recognize the ports:

   Pipe or HVAC nozzles - cylindrical protrusion

   Foundation ports - right triangular protrusion

   Electrical connections - square or rectangular protrusion

   a. In Solid Edge, click **Applications > Macros > Run Macro**.

   b. Run the [*Product Folder*]\Equipment\Client\Bin\SEDefinePort.exe macro.

   c. Follow the prompts to select and define each port.

Create Smart 3D reference data for use with Solid Edge components

All Smart 3D reference data is contained within the Catalog database. You create and modify reference data by using Excel spreadsheets. Then, use the **Bulkload Reference Data** application to post those changes from the spreadsheets to the Catalog database.

CustomInterfaces Worksheet

SymbolParameter

Specifies the parametric variable name as defined in the Solid Edge part. List all parameters that to be controlled by Smart 3D. You can reuse parameters from other parts if they all have the same type. You must add any parameters that do not already exist.

ReadOnly

Indicates whether you can modify this property.

1 - True. Indicates that the property cannot be modified.

0 - False. Indicates that the property can be modified.

OnPropertyPage

Indicates whether the attribute displays on the **Properties** dialog box for the equipment item

1 - True. Indicates that the attribute displays on the **Properties** dialog box.

0 - False. Indicates that the attribute does not display on the **Properties** dialog box.

PrimaryUnits

Specifies the expected unit of measure for this parameter.

UnitsType

Specifies the type of parameter that you are entering. For example, **Distance**, **Area**, **Angle**, and so on.

Type

Specifies the internal database storage type for the parameter. This is typically **Double** for non-whole numbers or **Integer** for whole numbers, but other values can be required for particular cases.

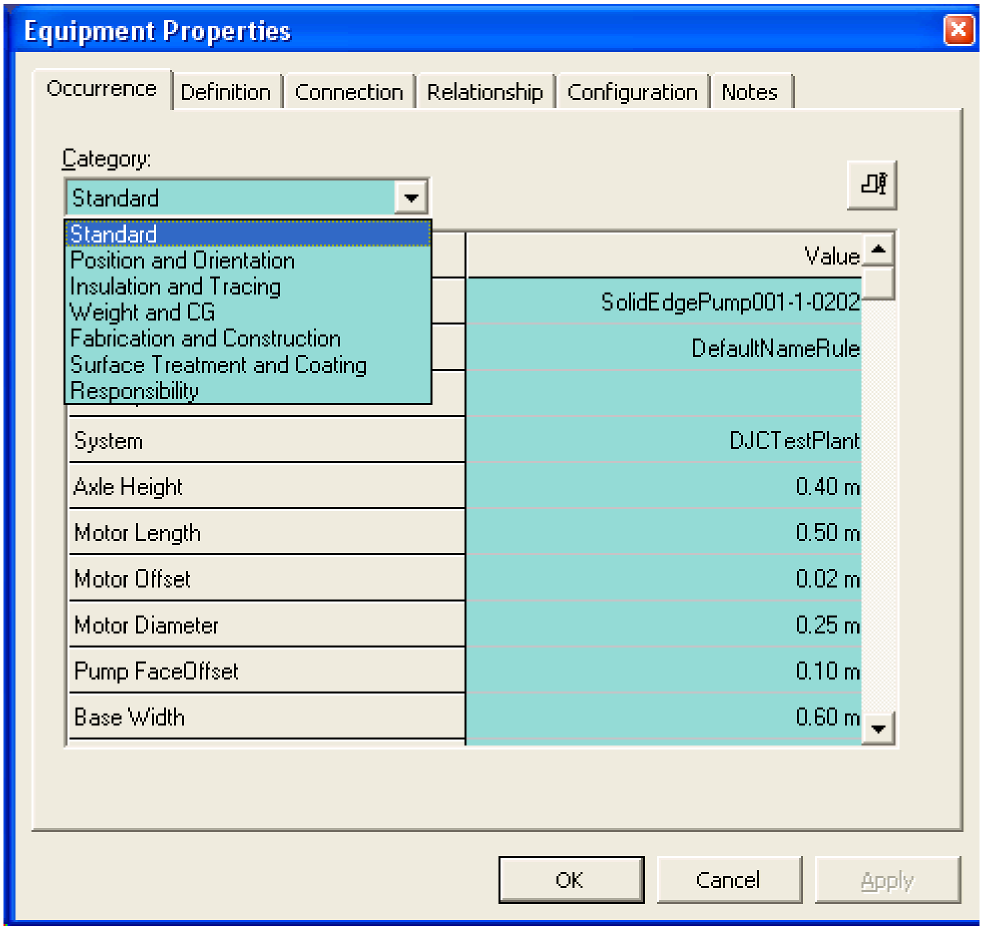
AttributeUserName

Specifies the name of the parameter as displayed in Smart 3D. Spaces are allowed.

CategoryName

Specifies the property category on which the variable displays in Smart 3D. If you leave this

field blank, Smart 3D displays the variable in the **Standard** category.



InterfaceName

Specifies the interface name for the object. Smart 3D retrieves object properties through interfaces. You can reuse properties of existing interfaces for any number of equipment types. You can add attributes to existing interfaces. If necessary, you can define new interfaces with new attribute properties. After you create an interface or attribute, you cannot delete or modify it.

Part Class Definition Worksheet

This worksheet is named with the part class, such as SolidEdgePump.

Definition

This row specifies the definition of all the attributes used by the equipment component. The following columns in the **Definition** row are required:

- **PartClassType**
- **SymbolDefinition**
- **UserClassName**
- **OccClassName**

- Additional attributes required for each equipment type

PartClassType

Specifies the type of Smart 3D object that you are defining. For an equipment class, this is **EquipmentClass**.

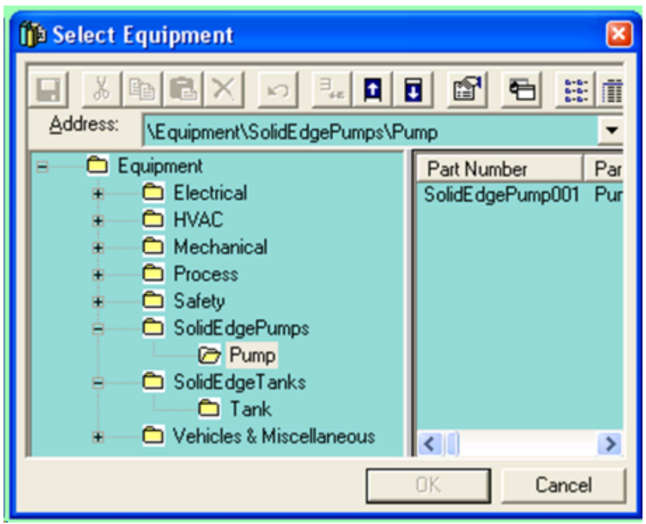
SymbolDefinition

Specifies the path to the Solid Edge assembly file in the symbols share for the Smart 3D project. Use the following format:

`SE2GSCAD.SEsymbol|SolidEdgeParts\[File Name].asm`

UserClassName

Specifies the class name displayed in Smart 3D. Spaces are allowed. In the following example, the labels **Pump** and **Tank** come from this column.



OccClassName

Specifies the class name that Smart 3D uses internally for the equipment class. You cannot use spaces in this column.

Additional required attributes

- Add a column in the **Definition** row for each parameter used from the **CustomInterfaces** worksheet.
- Type the Solid Edge variable enclosed in angle brackets (<>) after each **AttributeName**.
- Precede **AttributeName** with the **InterfaceName** for any attributes that occur in more than one interface. Use the following format:
- `InterfaceName::AttributeName<SEParameterName>`
- To make an attribute modifiable from within Smart 3D, precede that attribute with **OA:**. This makes the attribute an occurrence attribute. Use the following format:
- `OA:InterfaceName::AttributeName<SEParameterName>`

NOTE Port attributes, such as size, rating, and end preparation cannot be

occurrence attributes.

- Define **NozzleType** and **Nozzle(n):ID** for ports (nozzles). These values must match the definitions from Solid Edge.

Head

This row sets the order in which you enter data for individual part components. It also adds values for standard items such as nozzles and generic part data such as weight and center of gravity.

Start and **End**

These rows indicate the extent of the part data for individual components

R-Hierarchy Worksheet

This worksheet describes the hierarchy of the folders in the Equipment node of the Catalog.

RelationSource

Defines the name of a parent folder. These folders must be traceable through **RelationDestination** back to the **CatalogRoot** value.

RelationDestination

Defines the name of a child folder placed under the folder described in **RelationSource**.

ClassNodeType Worksheet

This worksheet defines the names displayed in Smart 3D.

ObjectName

Specifies the internal Smart 3D name for a folder. Smart 3D does not display this name.

Name

Specifies the name for **ObjectName**. Smart 3D does display this name.

Load and revise Smart 3D reference data

Use **Bulkload Reference Data** to load the completed Smart 3D reference data workbook into the Smart 3D Catalog database.

1. Click **Start > All Programs > Intergraph Smart 3D > Database Tools > Bulkload Reference Data**.

   *The **Bulkload** dialog box displays.*

2. Click **Add** to locate the Excel workbook.

3. Specify the **Bulkload mode** to use to load the workbook.

   Append to existing catalog - Adds any new records from the workbook that do not currently exist in the database. This option does not modify or delete any existing data in the database. Use this option when you are initially loading data into the Catalog.

   Add, modify, or delete records in existing catalog - Adds, modifies, or deletes records from the database that have been marked for action in the workbook. This option does not act on any records that are not marked for action in the workbook. Use this option to add,

modify, or delete data on a row-by-row basis. Type **A**, **M**, or **D** in the first column of each row to change in the database.

- **A** - Adds the contents of the row to the Catalog if it does not currently exist.
- **D** - Deletes the contents of the row from the Catalog, if possible.
- **M** - Modifies the row to match the current contents of the worksheet, if possible.

Create flavors - Creates a cached copy of each size of a component in the Catalog database. If you select this option, you are not required to have Solid Edge loaded on your computer to place Solid Edge components that exist in the Catalog. If you do not have Solid Edge loaded on your computer, you cannot modify Solid Edge components regardless of whether or not they are flavors.

*Depending on the selected **Bulkload mode**, the dialog box displays the appropriate boxes.*

4. Complete the rest of the boxes on the dialog box, and click **Load**.

Place and modify Solid Edge components in Smart 3D

You place Solid Edge equipment in Smart 3D the same way you place any other equipment item. If the Solid Edge equipment was loaded as a flavor, Smart 3D places the equipment item directly from the model cache. In this case, Solid Edge does not start. If the Solid Edge item was not loaded as a flavor, then Solid Edge starts and closes during the placement of the part. If you modify a Solid Edge item, Solid Edge opens and closes.

Smart 3D caches symbols in the model when you place them. That way, Smart 3D only needs to store each type of equipment in memory once. If you use Solid Edge to modify the symbol, you must flush the cache to get the latest version of the Solid Edge assembly file. Use the **Update Symbol** custom command to perform this operation.

1. Click **Tools > Custom Commands**.

   *The **Custom Commands** dialog box displays.*

2. Select **Update Symbol** in the **Command names** list.

   If **Update Symbol** does not display in the list, do the following:

   a. Click **Add**.

      *The **Edit Custom Command** dialog box displays.*

   b. Type `SymbolTestCmds.CUpdateSymboldefinition` in the **Command Progid** box.

   c. Type `Update Symbol` in the **Command name** box.

   d. Click **OK**.

3. Click **Run**.

   *The **Update** dialog box displays.*

4. Click **Select from Combo Box**.
5. Select the symbol from the **Symbol Definition Name** list.
6. Click **OK** or **Apply**.

Smart 3D updates the symbol definition to the latest version of the Solid Edge assembly file.

SECTION 4

Troubleshooting Symbols

While unlikely, symbols placed in a model can become corrupted or have problems. This section describes how to test symbols, what can cause symbols to become corrupt, and what you can do to fix corrupt symbols.

In addition to the symbols delivered with the software, Intergraph provides symbols and symbol fixes on the *Intergraph Smart Support* (*https://smartsupport.intergraph.com*) web site. These symbols are available on the product page under **Downloads > Smart 3D > Content**.

In This Section

Debugging Symbols with .NET

Use Microsoft Visual Studio debugging tools to debug symbols. You must have the latest Programming Resources software and Microsoft Visual Studio installed on the computer. For information on how to install the Programming Resources, please refer to the *Smart 3D Installation Guide*.

Setup

Add the following folders to your PATH environment variable:

- C:\Program Files\Smart3D\Core\Runtime
- C:\Program Files\Smart3D\GeometryTopology\Runtime

Preparing to Debug a Symbol

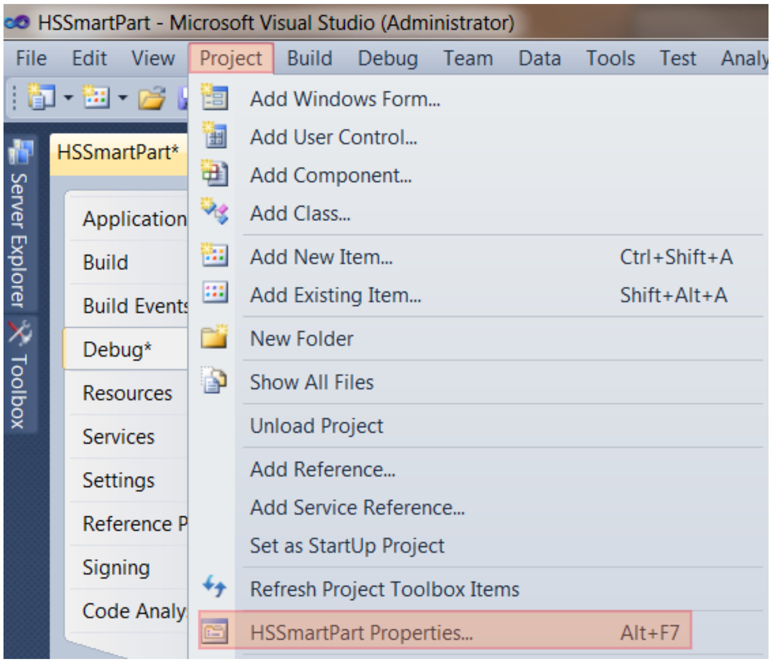
Before you can debug a symbol, you must ensure that there is entry for that symbol in the **CustomSymbolConfig.xml** or **SystemSymbolConfig.xml** file.

The **Update Custom Symbol Configuration (UCSC)** command looks at the symbols in the SharedContent folder and makes entries in the SymbolConfig.xml files. Entries for symbols that you have created in the SharedContent\Custom Symbols folder are added to the **SharedContent\Xml\CustomSymbolConfig.xml** file. Entries for symbols supplied by Intergraph are added to the **SharedContent\Xml\SystemSymbolConfig.xml** file.

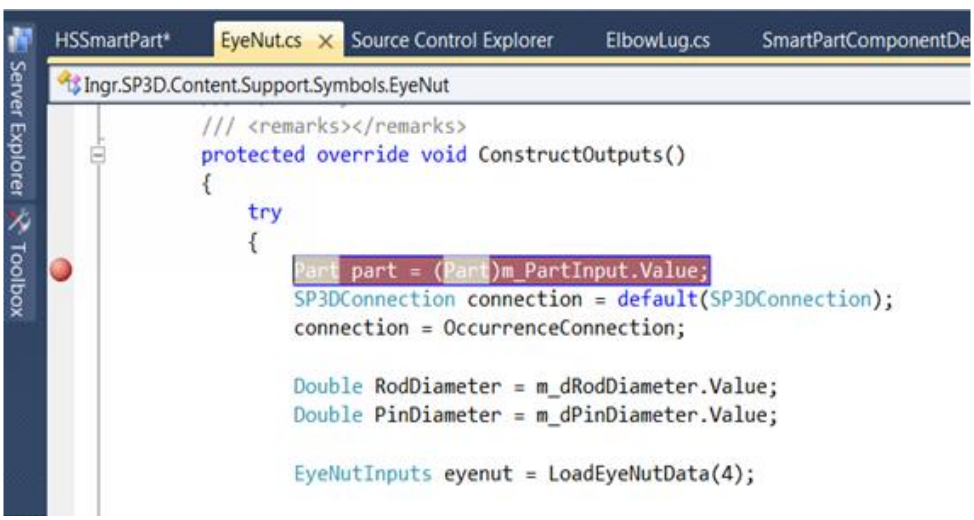
Debugging a Symbol .NET Project

1. In .NET, open the project to debug.

2. Click **Project** and select the project you want to debug. For example, in the following image select **HSSmartPart Properties (Alt+7)**.



3. Click **Debug** and select **Start external program**.
4. Click Browse and navigate to **S3DHost.exe**.
5. Open the code page for the symbol and add break points.



6. Press **F5** to run the project.

Smart 3D opens. Create a new session file or open an existing session file and place the symbol. The control is passed to .NET at the break point and the normal .NET debug commands such as Step Into and Step Over can be used.

★ **IMPORTANT** When symbols are placed for the first time in the model, a cache is created in the Model database and the actual symbol code will not run a second time or beyond. Please

refer to *Edit Symbol Occurrence* (on page 48) for information on how to force the execution of symbol code for debugging purposes.

See Also
Testing Symbols (on page 47)

Testing Symbols

Two custom commands are delivered with the software to help symbol designers:

- Locate an existing symbol and change the inputs. During the design phase of symbol creation, it can be very time consuming trying to use the full application to test a symbol, especially if it requires multiple bulkloading to the catalog. For more information, see *Edit Symbol Occurrence* (on page 48).
- Update a symbol definition from a list of symbol definitions in the active connection, or update an object given an Object ID (Database ID) and an Interface ID. This issue can arise when the symbol is cached and you want to test a change in the code. If there already is an existing symbol available for the set of input parameters, then the changed symbol code will not run. For more information, see *Update Symbol* (on page 47).

See Also
Troubleshooting Symbols (on page 45)

Update Symbol

This utility calls the update mechanism on a symbol definition or other object so that the software will recalculate any symbols connected to the object.

⚠ CAUTIONS

- You must understand the consequences of trying to recalculate an object. Errors can occur when the context is incomplete in allowing one or more related objects to recalculate. This error can occur when one object is read-only or missing.
- A symbol definition may have thousands of symbols connected to it. Each symbol will recalculate if an update is called on the definition. This utility is not designed to handle it and should be used in small models.

Symbols Tab

Key in

Select this option to key in the symbol definition name to update in the **Symbol Definition Name**. Use this option if you have more than 10 to 15 symbols in the model.

Select from Combo Box

Select this option to select the symbol to update in the **Symbol Definition Name**. Use this option only if your model is very small, 10 to 15 symbols.

Symbol Definition Name

Displays all the symbol definitions available in the model from which you can select one to update. You can also type the symbol definition name to update, for example: SP3DOP3.COP3.

Apply or **OK**

Click to update the selected symbol definition and cause each symbol of that definition to recalculate.

Object Tab

ObjectID

Select the object to update.

InterfaceID

Select which interface on the selected object to update.

Apply or **OK**

Click to recalculate the selected object interface.

Workflow

1. Click **Tools > Custom Commands**.
2. Click **Add**.
3. In the **Command ProgID** box, type SymbolTestCmds.CUpdateSymbolDefinition
4. In the **Command name** box, type a name for the utility. We recommend you type *Update Symbol* or *Object Test Command* for the command name.
5. Click **OK**.
6. Select *Update Symbol or Object Test Command* from the list of command names, and then click **Run**.
7. Select the symbol or key in the symbol to update.
8. Click **Apply**.

See Also

Testing Symbols (on page 47)

Edit Symbol Occurrence

This utility edits an existing symbol occurrence in the model.

⚠ **CAUTION** This command assumes the person using it is the symbol designer who knows what the valid inputs for the symbol are. This command does not check input parameter values that you type as it cannot determine valid inputs for the symbol.

Options

Parameters

Displays all the input parameter of the selected symbol.

- Index - Displays the index number of the input parameter.
- Name - Displays the name of the input parameter.
- ByRef - Indicates if the parameter is passed by a reference.
- Value - Type a value for the parameter.

Graphics

Displays the graphic elements that are inputs for the selected symbol.

Representation

Displays the display aspects that the symbol supports.

Workflow

1. Click **Tools > Custom Commands**.
2. Click **Add**.
3. In the **Command ProgID** box, type **SymbolTestCmds.CEditSymbolOccurence**.
4. In the **Command name** box, type a name for the utility. We recommend you type **Edit Symbol Occurrence** for the command name.
5. Click **OK**.
6. Select **Edit Symbol Occurrence** from the list of command names, and then click **Run**.
7. Select the symbol in the model.
8. Test the input parameters as needed.

See Also

Testing Symbols (on page 47)

Sources of Errors

Bulkloading

Symbols can be broken in the model because of an incorrect bulk load operation. The most common bulkloading mistakes are:

- Deleting the symbol definition, flavor manager, or flavor in the catalog when the symbol still exists in the model.
- Setting incorrect parameter values in the catalog. For example, setting a pipe diameter to be zero.

Synchronize Model with Catalog

After bulkloading is complete or if symbol definitions have been changed and the major version number of the definition increased, you must run the **Tools > Synchronize Model with Catalog** command in the Project Management task for all models that use the catalog or changed definitions.

Symbols Folder

The software expects to find the symbol DLL files in a single folder, usually located under the SharedContent folder. This symbols folder is specified when the catalog database is created. Doing any of the following can cause symbol problems:

- An incorrect symbols folder is specified when the catalog database is created.
- The symbols folder is moved after the catalog database is created.

- The catalog database is backed up and then restored to a different server, but the symbols folder is not copied to the new server.
- Using different custom symbol folders for the different clients of the server.

Usage of Cached/Non-cached Symbols

The default method is to cache symbols whenever possible. A symbol definition that has a non-parametric input (for example, a part) will not be cached even if all the other inputs are parameters. However, if a custom method (CMcache) is written to convert the part into a parameter, then the symbol will be cached.

To make this change from a non-cached to cached for the case where non-cached symbols have already been placed in the model, the major version number of the symbol definition must be increased and the **Tools > Synchronize Model with Catalog** command in the Project Management task run. If this is not done, then the change in the way the part input is treated results in an error as the symbols already placed in the model are expecting a part, and not a parameter, and will fail to compute.

Multiple Outputs with Same Name

A .NET symbol with duplicate output names is not allowed. When such a symbol is placed in the model, the transaction will be aborted. Check the Core error log file for errors. If there are existing .NET symbol occurrences that have duplicate output names, they cannot be recomputed. For information on how to resolve the error, please refer to the *Symbol has outputs with duplicate names* topic in *Smart 3D Database Integrity Guide.*

Software Updates

Errors can occur if the server and the client software are not the same software version. All the symbols must be the same version to guarantee compatibility. The best method for ensuring that the symbols are the same on the clients and the server is to use the symbol definition download feature by placing the symbols in CAB files. For more information, see Distributing Symbols Automatically.

See Also
Troubleshooting Symbols (on page 45)

Error Investigation Methods

For errors received on the definition:

- Check for incomplete or wrong definition.
- Check for wrong versions of a symbol definition.
- Check for input mismatches.
- Check for output mismatches.
- Check for properties mismatches.

For errors received on symbols:

- Check for unsynchronized data.
- Use the database integrity check in the Project Management task.

SECTION 5

Symbol Validation Tool

The **Symbol Validation Tool** verifies that symbols are defined in a valid manner. The tool identifies an invalid definition such as defining two outputs with the same name. You should also run the **Symbol Validation Tool** after you modify a symbol to verify that the new symbol is compatible with the old symbol.

The **Symbol Validation Tool** is delivered in [Product Folder]\ProjectMgmt\Tools\Bin. You can run the Symbol Validation Tool with or without Smart 3D, but if you want to validate symbol definitions in a model database, you must have Smart 3D.

What do you want to do?

- *Verify a single symbol definition* (on page 51)
- *Compare multiple symbol definitions* (on page 53)
- *Run comparisons from the command line* (on page 54)

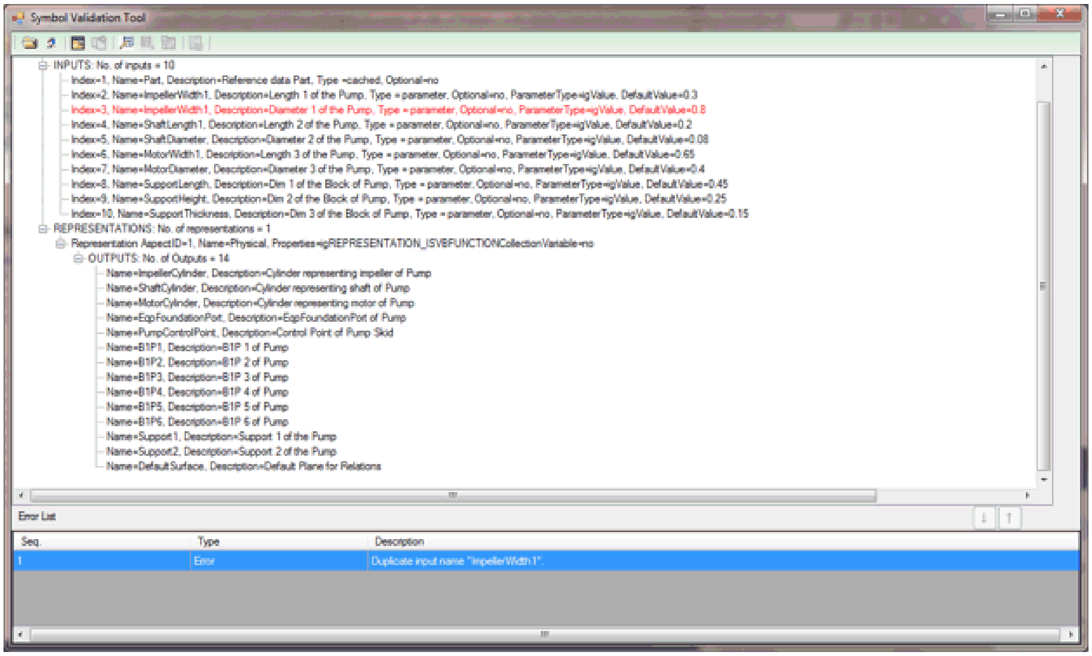
Verify a single symbol definition

1. Double-click [*Product Folder*]\ProjectMgmt\Tools\Bin\SymbolValidationTool.exe.
2. Click **Open Symbol** .
3. Select **Symbol definition in DB** if the symbol to check is in the database.

   -OR-

   Select **Dll or XML** if you have the symbol source files.
4. Define the database connection information, or select the DLL or XML file to open.
5. Select the **ProgID** to open, and then click **OK**.

The symbol definition displays.



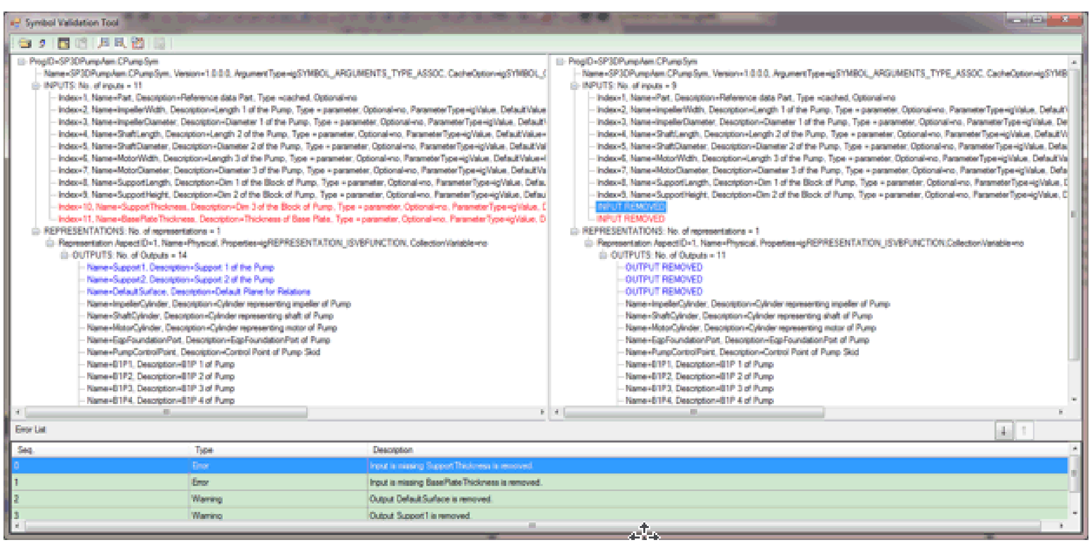
Any errors in the symbol display in the bottom list view.

6. Click **Compare with Symbol** .
7. Select **Symbol definition in DB** if the symbol to compare against is in the database.

   -OR-

   Select **Dll or XML** if you have the symbol source files.
8. Define the database connection information, or select the DLL or XML file to open.
9. Select the **ProgID** to open, and then click **OK**.

   The symbol definition displays.



The tool highlights differences between the two symbol definitions. The left pane is the basis of the comparison. The right pane is the symbol that the tool is validating. The bottom list view displays all the differences that were found.

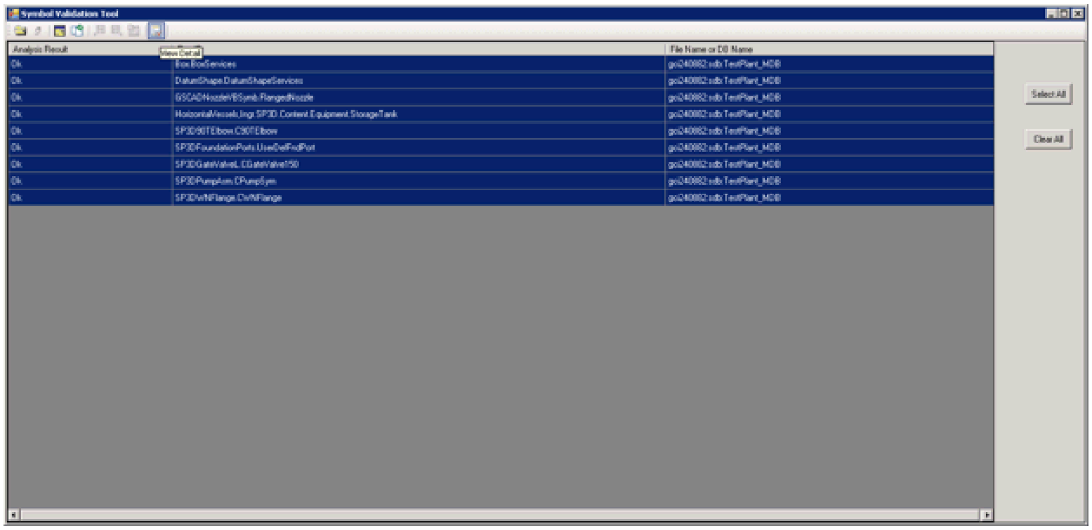
Compare multiple symbol definitions

1. Click **Open multiple symbols** .
2. Select **Symbols from model database** if the symbols to check are in the database.

   -OR-

   Select **Open symbol directory** if you have the symbol source files.

   The tool pulls and validates all symbol definitions from the model database or from the defined folder, including sub folders.



3. Select one or more symbols from the list, and then click **View Details**
4. Click **Compare with Multiple Symbols** to compare multiple symbols.

   The tool displays all the matching ProgIDs in the basis list and new list. If you have changed ProgIDs, use the ***ProgID Mapping File*** *box to enter the XML file name that contains the mapped names.*
5. Select from the displayed list of symbol definitions that were compared to view the details of the comparison results.

Run comparisons from the command line

You can also run the Symbol Validation Tool from the command line using a parameter file as an input argument. However, the command line option only compares list to list and generates a report. You cannot show the details of each symbol definition or define a ProgID mapping file.

Example 1: Compare files to files

Basis Symbols:
Directory:D:\SymbolComp\eCustomerDLLs
New Symbols:
Directory:D:\SymbolComp\NewDLLs
Report File:D:\test\sample.xlsx

Example 2: Compare database to files

Basis Symbols:
DBType:SQL
Server:122sqlserver
Site:Test_SDB
Model:Test_MDB
New Symbols:
Directory:D:\test
Report File:D:\test\sample.xlsx

SECTION 6

Exporting Symbols to IFC

There are specific content requirements when exporting footings or foundations to Industry Foundation Classes (IFC).

The software takes the set of GTyped surfaces in a symbol's output collection and uses them to construct a solid. If there are duplicate or missing surfaces, or if the construction of the solid fails, then the software cannot export the footing or the foundation.

One problem is that surfaces can have caps. Currently, the delivered symbols only create surfaces of projection, but Cones, Spheres, Tori, B-Spline Surfaces, Surfaces of Revolution, and Ruled Surfaces can also have caps. The presence or absence of caps is indicated with a Boolean parameter, but the capping planes are not created until needed (usually during locate) and are not normally persisted.

Some of the delivered symbols explicitly include the end caps in their output collection and set the capped parameter to False. Other symbols set the capped parameter to True and do not add the capping planes to the output collection. Finally, some symbols have capped surfaces and include the capping planes.

To accommodate all of the symbols currently delivered, the following criteria is checked during IFC export:

1. If the surface does not set its Boolean "Has Caps" parameter to True, no capping planes are created and all surfaces in the output collection are expected to enclose a single void.
2. If the capped parameter is set to True, but there are other surfaces in the output collection, the software assumes that all of the surfaces in the output collection enclose a void and no capping planes are created.
3. If the output collection contains a single surface and its capped parameter is set to True, then capping planes are created for the purpose of creating the solid but are not persisted.

S E C T I O N 7

Structure Symbols

The software delivers several basic structure symbols for stairs, ladders, handrails, footings, and equipment foundations. For information about member cross-sections symbols, refer to the *2D Symbols User's Guide* or the *Structure Reference Data Guide*.

In addition to the symbols delivered with the software, you can create your own symbols that you can use in your model. For more information about creating symbols, refer to the *Smart 3D Reference Data Guide*, available from the **Help > Printable Guides** command in the software.

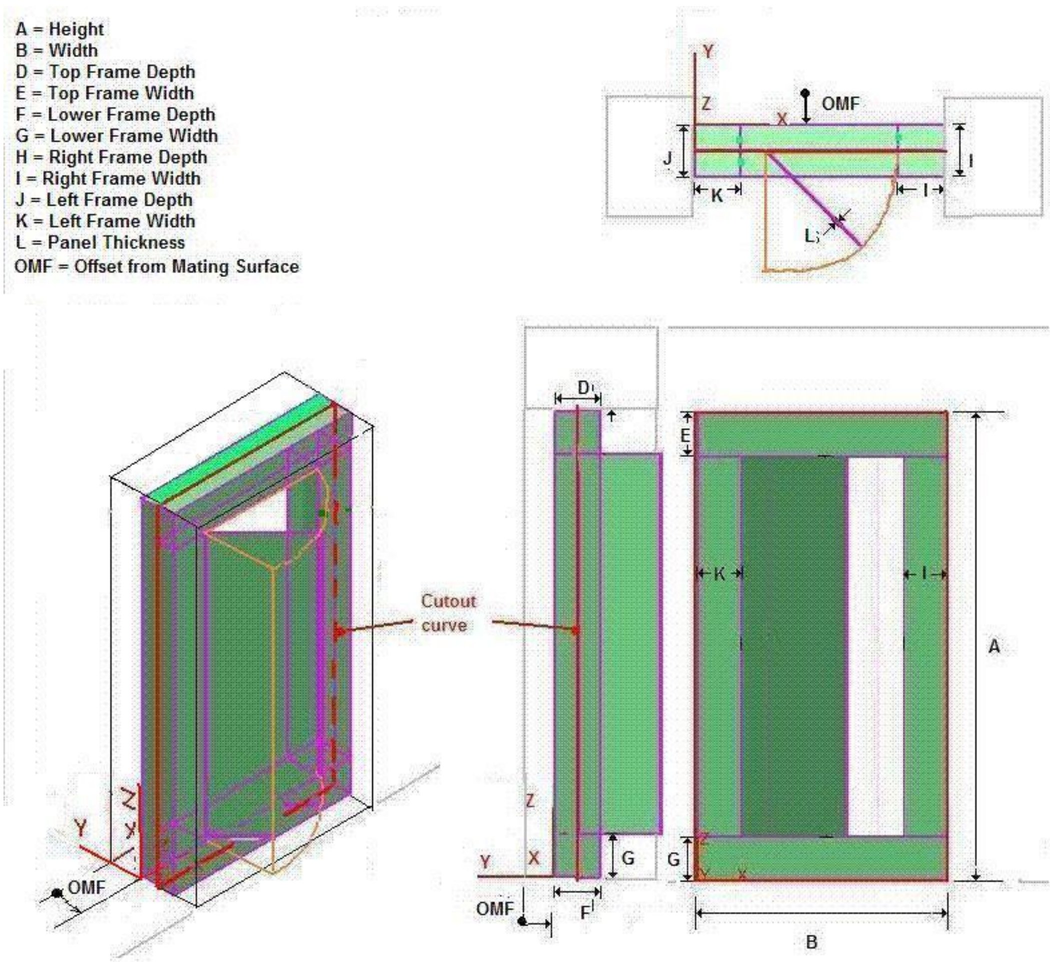
Doors and Windows

Topics

SimpleDoor.Asm

Description: Doors and frames
Symbol Name: SimpleDoorAsm.SimpleDoor_1_Sym

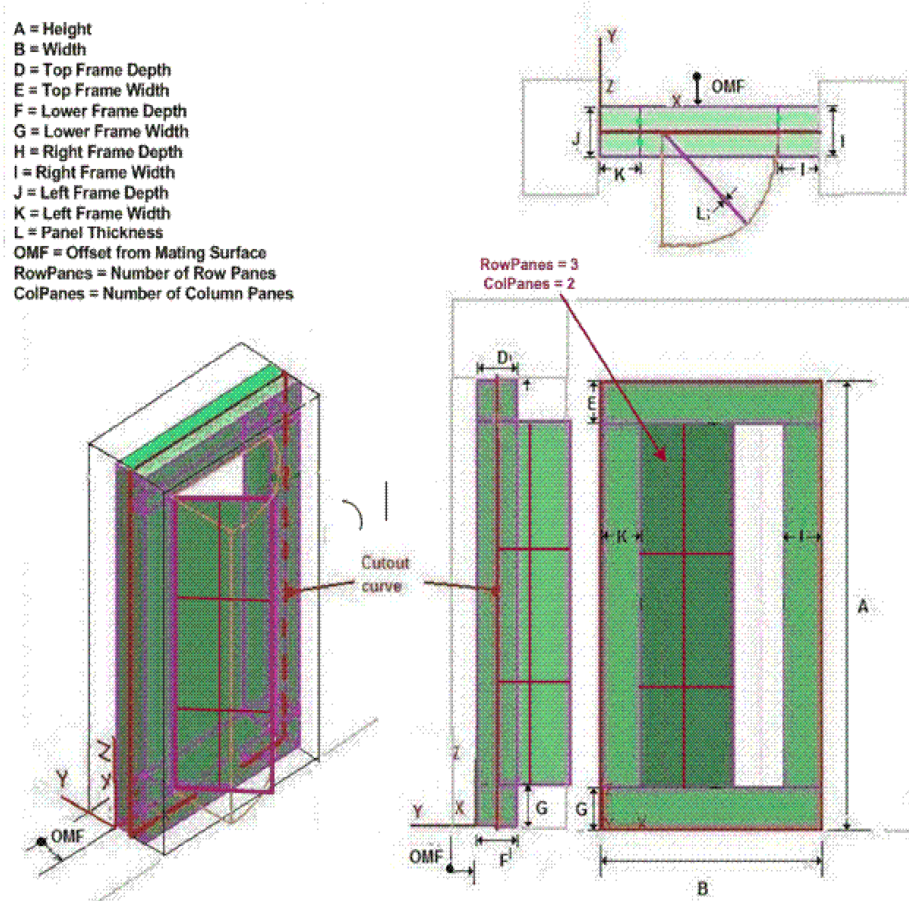
Workbook: SimpleDoor.xls
Workbook Sheet: SimpleDoorAsm



SimpleWindowAsm

Description: Windows and frames
Symbol Name: SimpleDoorAsm.SimpleWindow_1_Sym

Workbook: SimpleDoor.xls
Workbook Sheet: SimpleWindowAsm

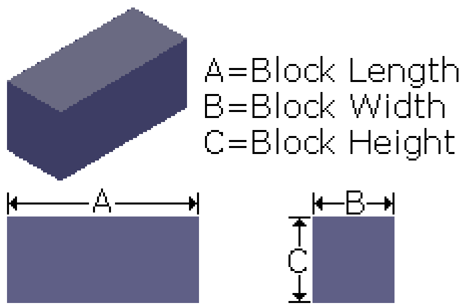


Equipment Foundations

Topics

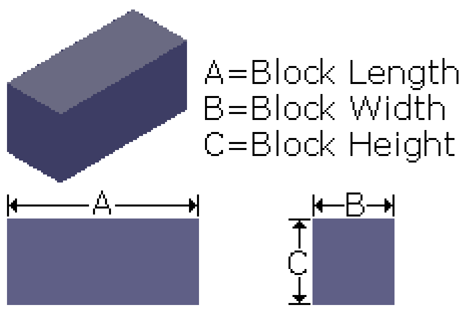
SPSEqpFndMacros.BlockFndAsmDef

Description: equipment block foundations
Symbol Name: SPSEqpFndMacros.BlockFndAsmDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: BlockFndnAsm



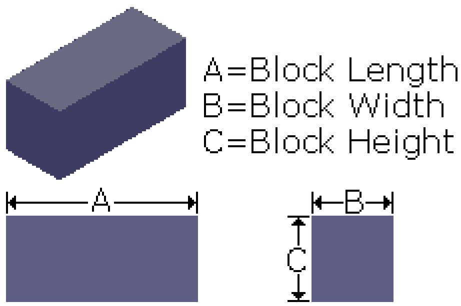
SPSEqpFndMacros.BlockFndCompDef

Description: equipment block foundations
Symbol Name: SPSEqpFndMacros.BlockFndCompDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: BlockFndnComp



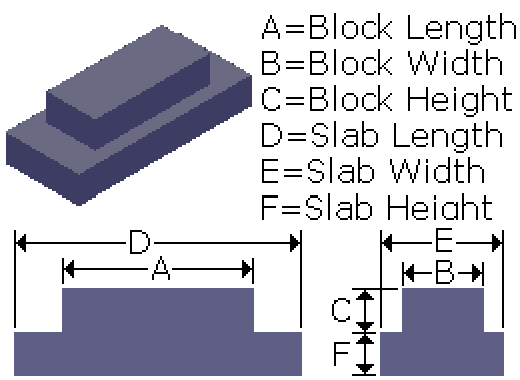
SPSEqpFndMacros.BlockFndDef

Description: equipment block foundations
Symbol Name: SPSEqpFndMacros.BlockFndDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: BlockFndn



SPSEqpFndMacros.BlockSlabFndAsmDef

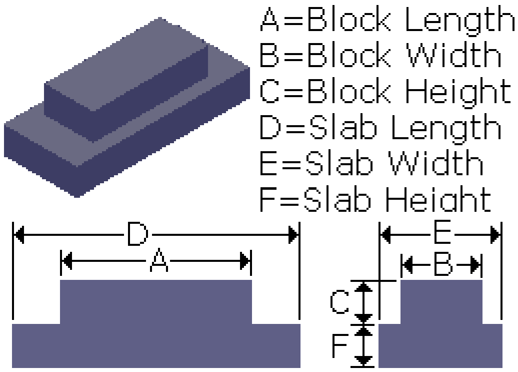
Description: equipment block foundations
Symbol Name: SPSEqpFndMacros.BlockSlabFndAsmDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: BlockAndSlabFndnAsm



SPSEqpFndMacros.BlockSlabFndDef

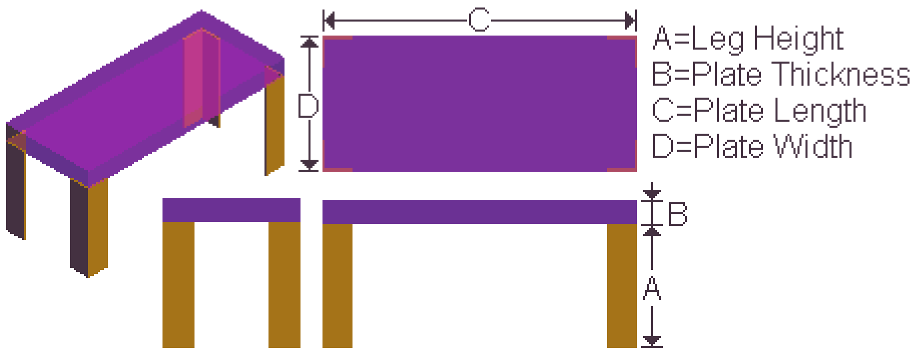
Description: equipment block foundations
Symbol Name: SPSEqpFndMacros.BlockSlabFndDef

Workbook: StructEquipFoundations.xls
Workbook Sheet: BlockSlabFndn



SPSEqpFndMacros.FrameFndAsmDef

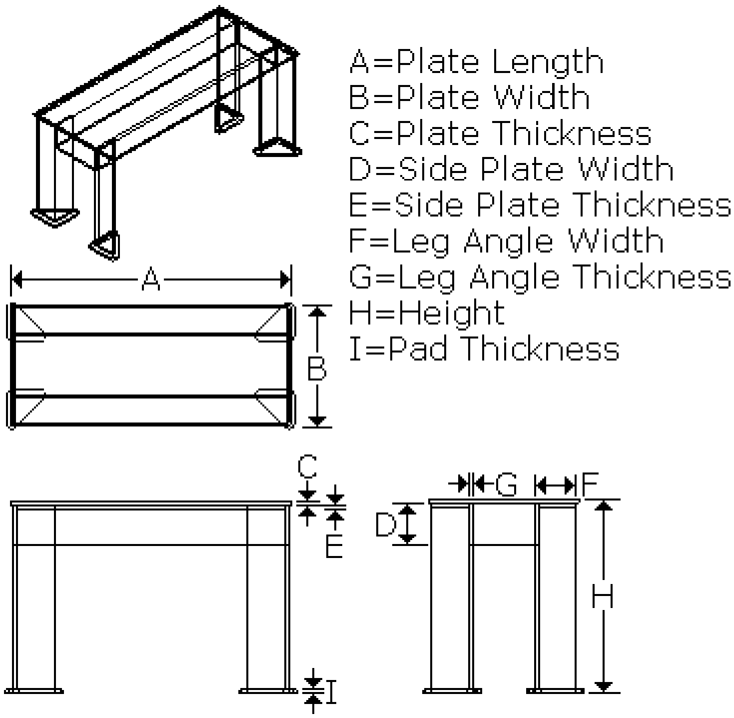
Description: equipment frame foundations
Symbol Name: SPSEqpFndMacros.FrameFndAsmDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: FrameFndnAsm



SPSEqpFndMacros.FrameFndDef

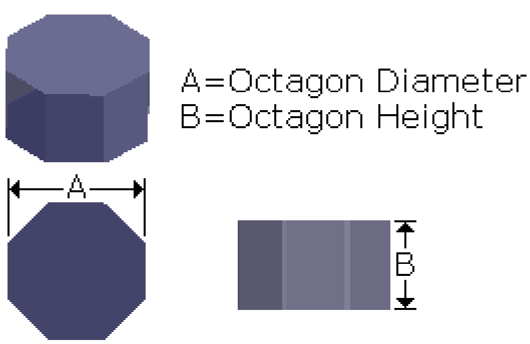
Description: equipment frame foundations
Symbol Name: SPSEqpFndMacros.FrameFndDef

Workbook: StructEquipFoundations.xls
Workbook Sheet: LegAngleFrameFndn



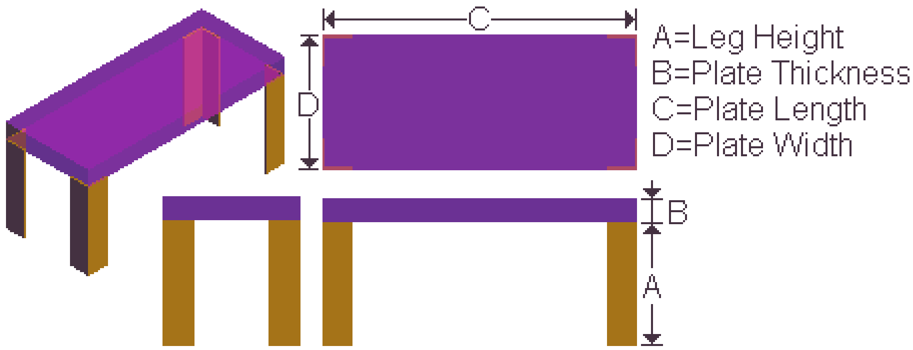
SPSEqpFndMacros.OctagonFndDef

Description: equipment frame foundations
Symbol Name: SPSEqpFndMacros.OctagonFndDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: OctagonFndn



SPSEqpFndMemSys.FrameFndnAsmWMemSysDef

Description: equipment frame foundations with member systems definitions
Symbol Name: SPSEqpFndMemSys.FrameFndnAsmWMemSysDef
Workbook: StructEquipFoundations.xls
Workbook Sheet: FrameFndnAsmWMemSys



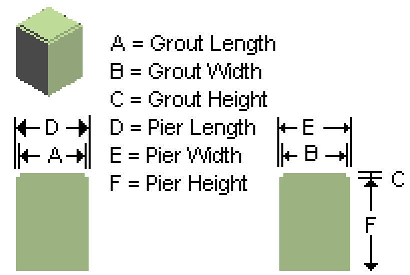
Footings

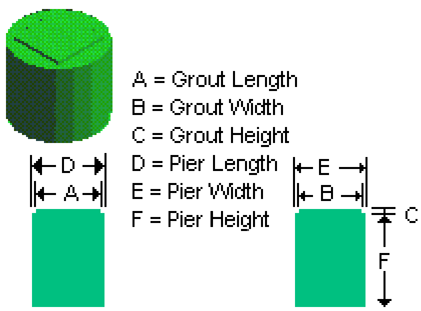
Topics

SPSFootingMacros.BoundedPierFtgAsmDef

Description: rectangular and circular pier footings
Symbol Name: SPSFootingMacros.BoundedPierFtgAsmDef

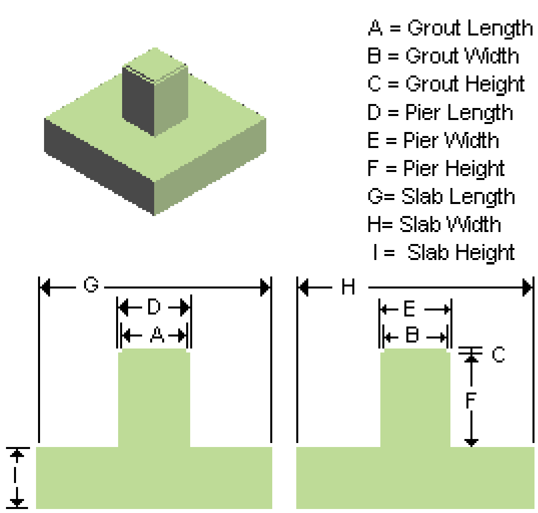
Workbook: StructFootings.xls
Workbook Sheet: PierFootingAsm





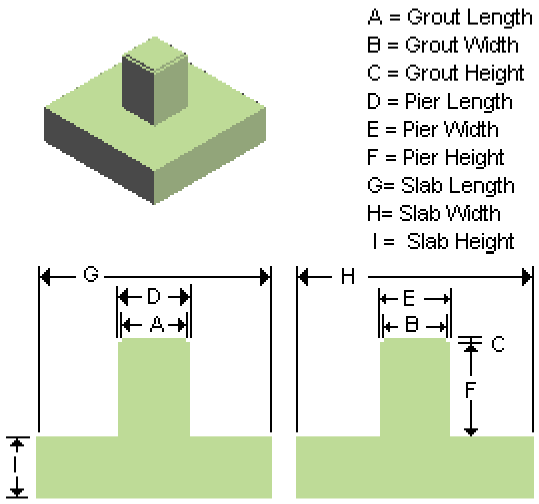
SPSFootingMacros.FtgGroutPadSym

Description: pier and slab footings
Symbol Name: SPSFootingMacros.FtgGroutPadSym
Workbook: StructFootings.xls
Workbook Sheet: FootingGroutPad



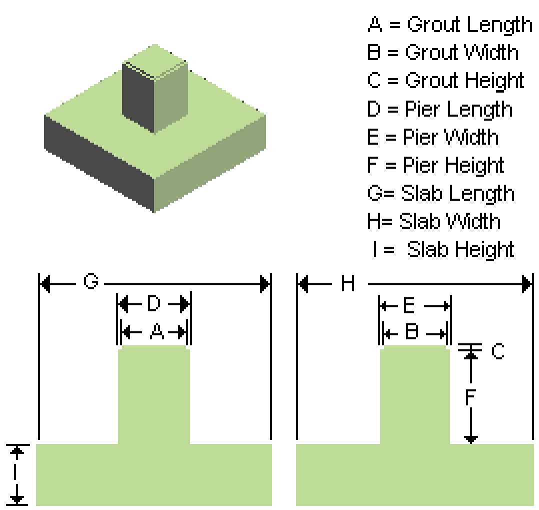
SPSFootingMacros.FtgPierSym

Description: pier and slab footings
Symbol Name: SPSFootingMacros.FtgPierSym
Workbook: StructFootings.xls
Workbook Sheet: FootingPier



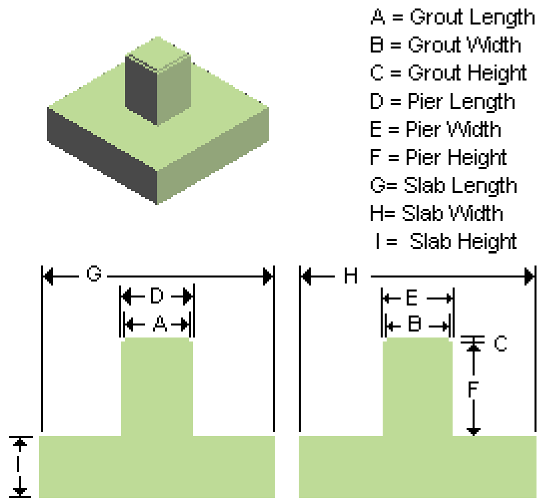
SPSFootingMacros.FtgSlabSym

Description: rectangular and circular slab footings
Symbol Name: SPSFootingMacros.FtgSlabSym
Workbook: StructFootings.xls
Workbook Sheet: FootingSlab

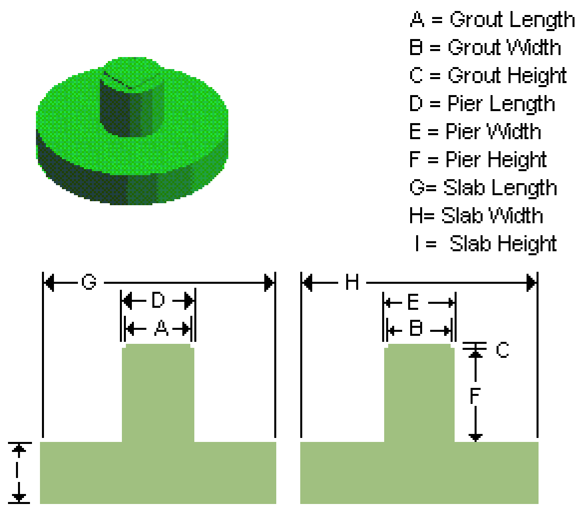


SPSFootingMacros.PierAndSlabFtgAsmDef

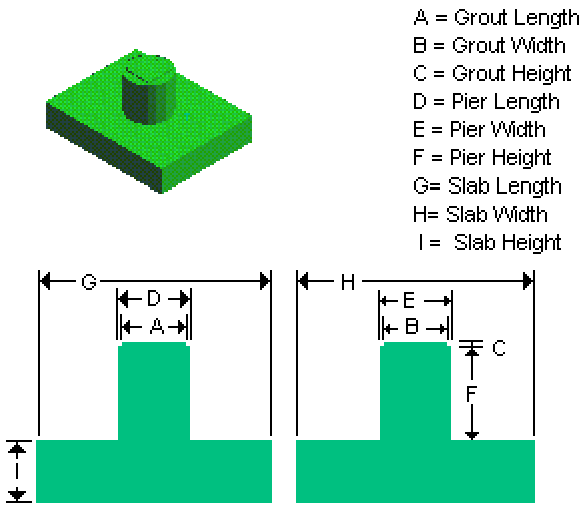
Description: pier and slab footings
Symbol Name: SPSFootingMacros.PierAndSlabFtgAsmDef
Workbook: StructFootings.xls
Workbook Sheet: PierAndSlabFootingAsm



Rectangular pier and slab footing



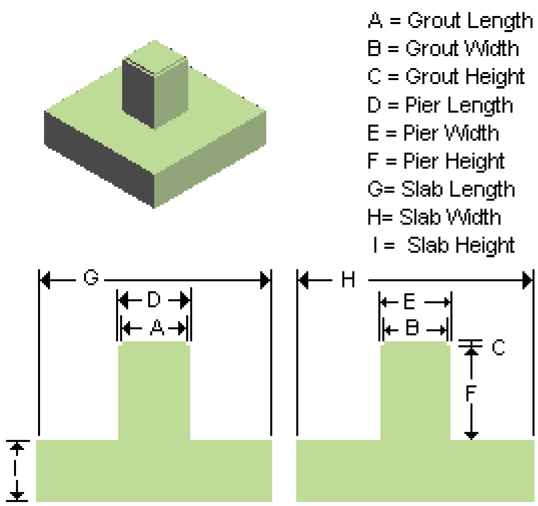
Circular pier and slab footing



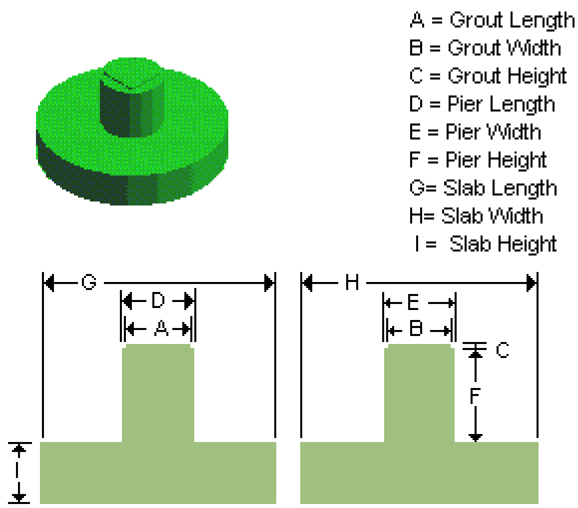
Circular pier and rectangular footing

SPSFootingMacros.PierAndSlabFtgSym

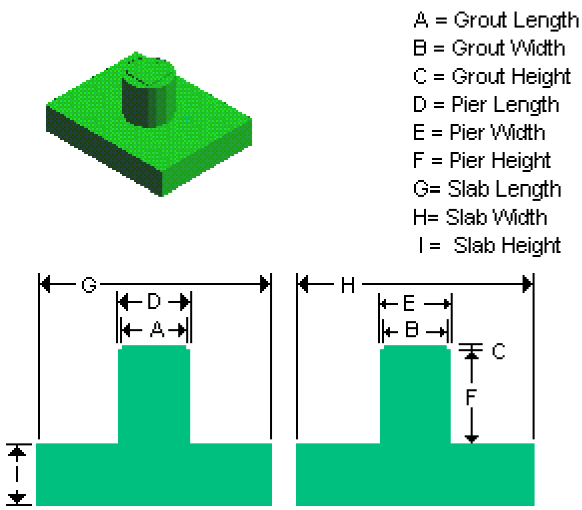
Description: pier and slab footings
Symbol Name: SPSFootingMacros.PierAndSlabFtgSym
Workbook: StructFootings.xls
Workbook Sheet: PierAndSlabFooting



Rectangular pier and slab footing



Circular pier and slab footing

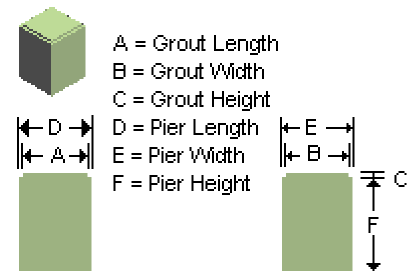


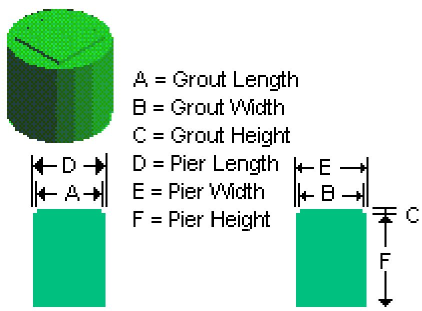
Circular pier and rectangular footing

SPSFootingMacros.PierFtgAsmDef

Description: rectangular and circular pier footings
Symbol Name: SPSFootingMacros.PierFtgAsmDef

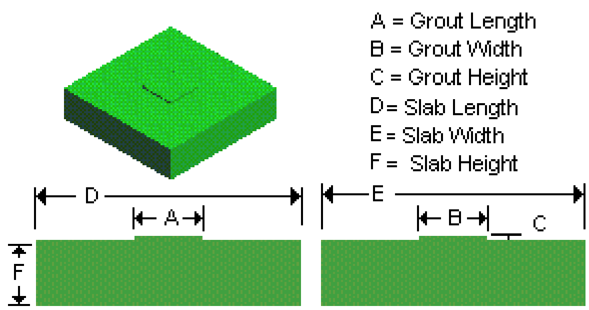
Workbook: StructFootings.xls
Workbook Sheet: PierFootingAsm

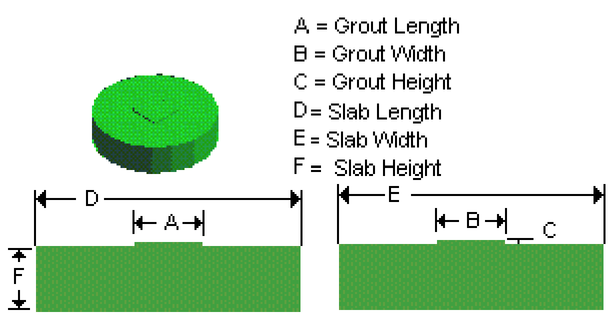




SPSFootingMacros.SlabFtgAsmDef

Description: rectangular and circular slab footings
Symbol Name: SPSFootingMacros.SlabFtgAsmDef
Workbook: StructFootings.xls
Workbook Sheet: SlabFootingAsm



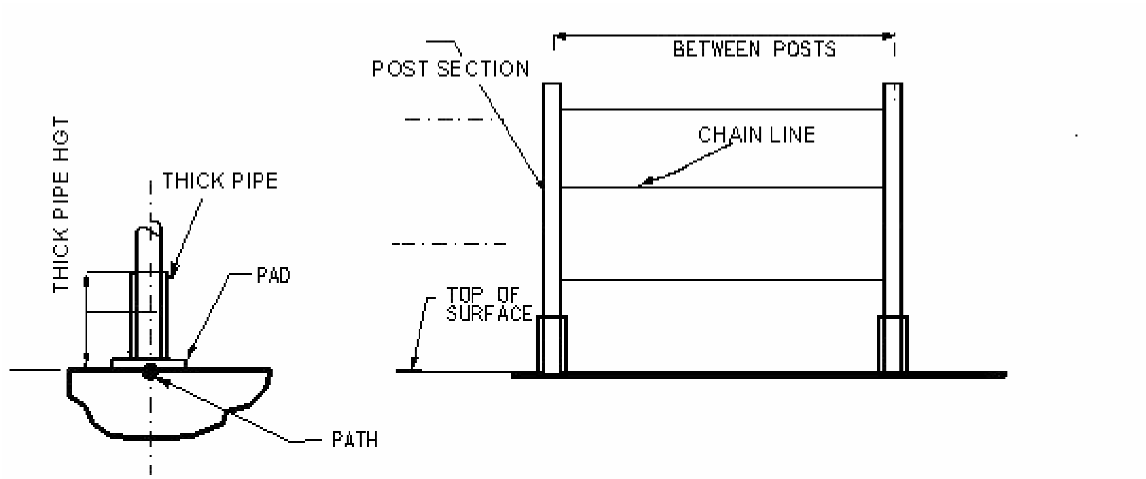


Handrails

Topics

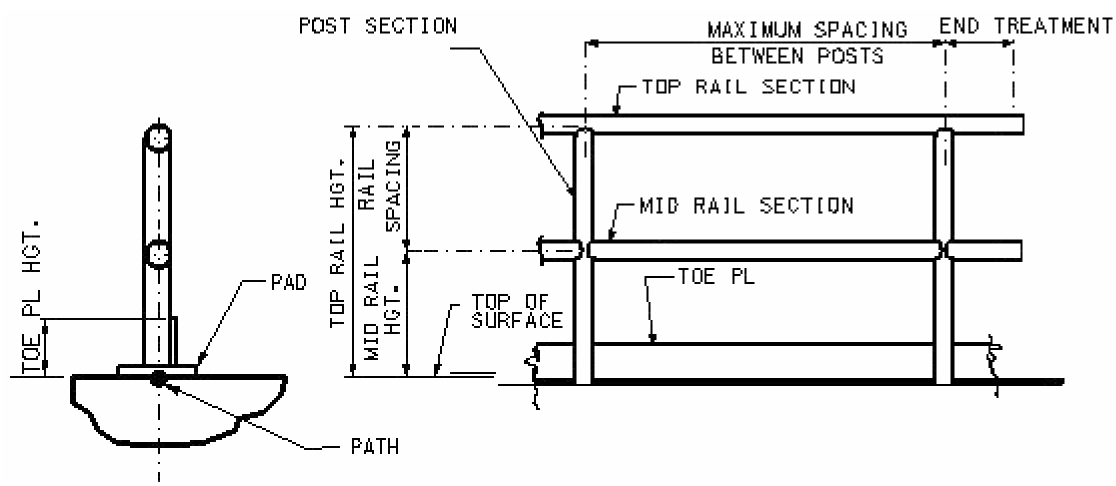
SPSHandrail.ChainRailing

Description: chain hand railing
Symbol Name: SPSHandrail.ChainRailing
Workbook:
Workbook Sheet:
Prog ID: SPSHandrail.ChainRailing



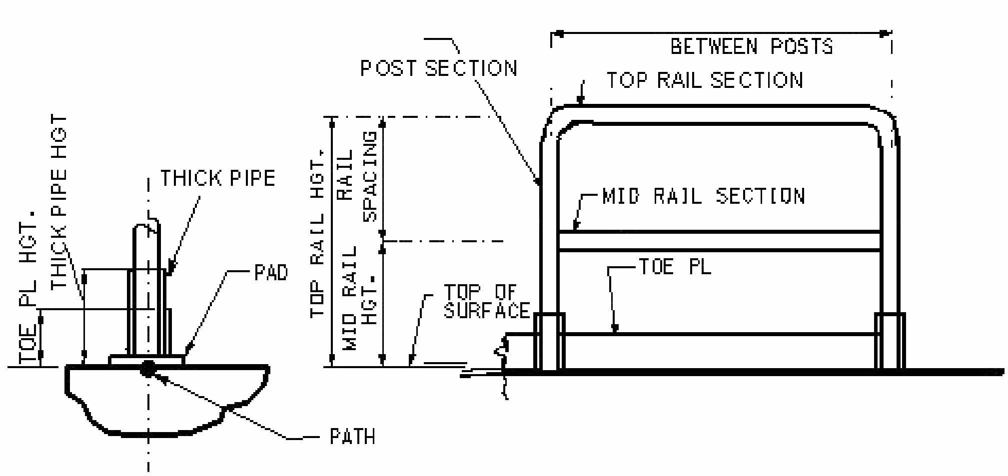
SPSHandrail.FixedHandrail

Description: fixed hand railing
Symbol Name: SPSHandrail.FixedHandrail
Workbook:
Workbook Sheet:
Prog ID: SPSHandrail.FixedHandrail



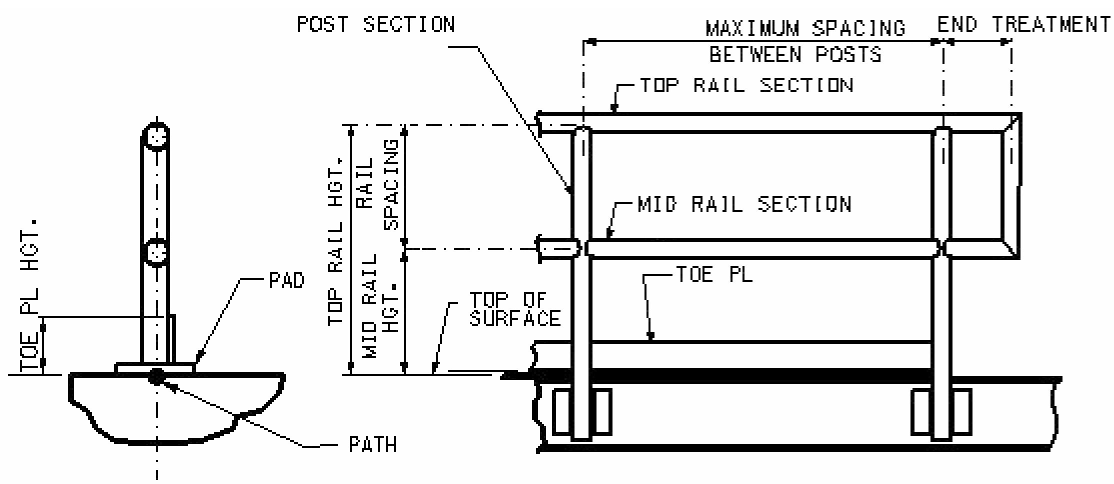
SPSHandrail.RemovableHandrail

Description: removable hand railing
Symbol Name: SPSHandrail.RemovableHandrail
Workbook:
Workbook Sheet:
Prog ID: SPSHandrail.RemovableHandrail



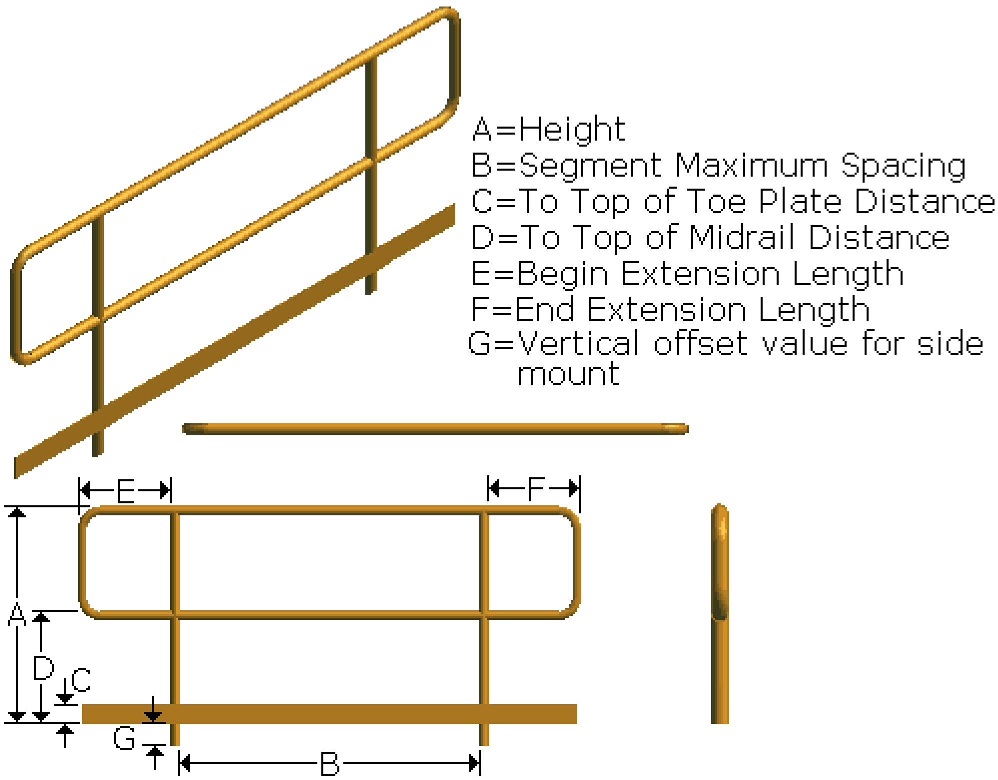
SPSHandrailMacros.TypeA

Description: side-mounted handrail
Symbol Name: SPSHandrailMacros.TypeA
Workbook: StructHandrails.xls
Workbook Sheet: HandrailTypeA



SPSHandrailMacros.TypeASideMount

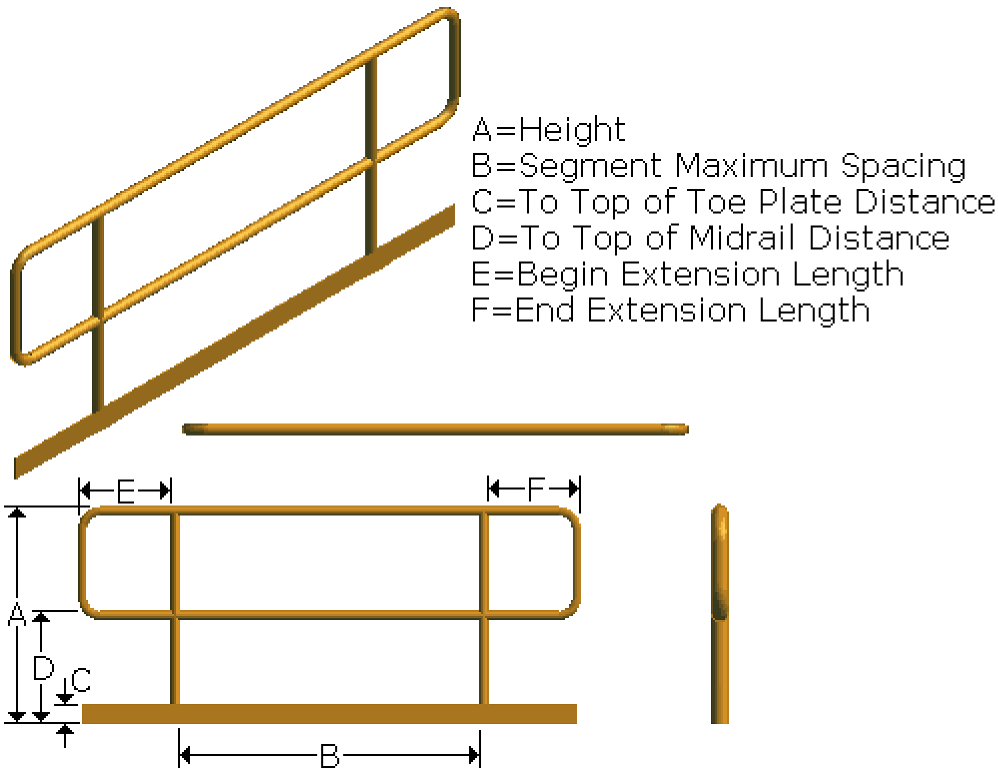
Description: side-mounted handrail
Symbol Name: SPSHandrailMacros.TypeASideMount
Workbook: StructHandrails.xls
Workbook Sheet: HandrailTypeA_SideMounted



SPSHandrailMacros.TypeATopEmbedded

Description: top-mounted handrail embedded in surface
Symbol Name: SPSHandrailMacros.TypeATopEmbedded

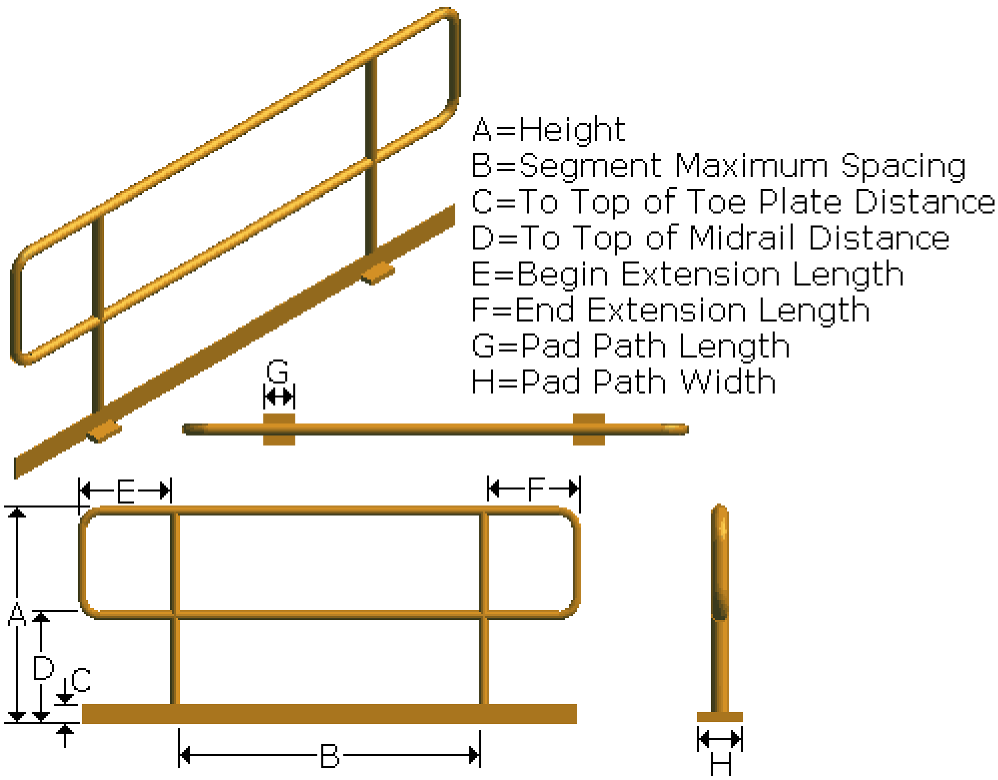
Workbook: StructHandrails.xls
Workbook Sheet: HandrailTypeA_TopEmbedded



SPSHandrailMacros.TypeATopMounted

Description: handrail top mounted on pad
Symbol Name: SPSHandrailMacros.TypeATopMounted

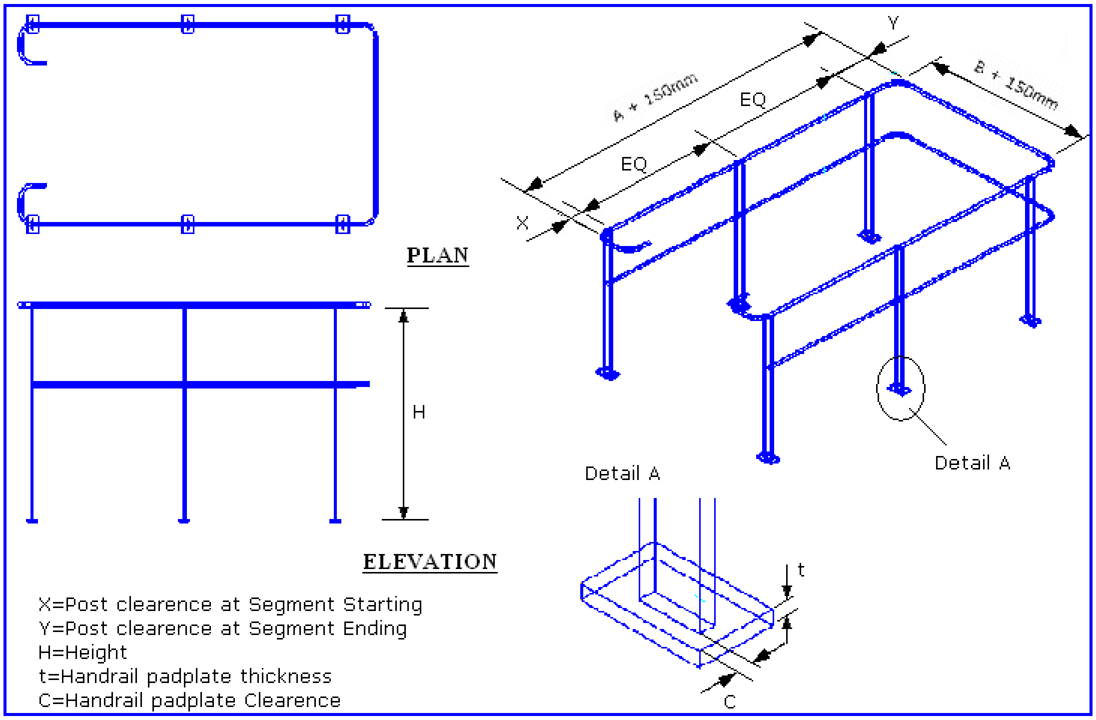
Workbook: StructHandrails.xls
Workbook Sheet: HandrailTypeA_TopMounted



SPSLadderOpeningHR.LadderOpeningHR

Description: ladder opening handrail
Symbol Name: SPSLadderOpeningHR.LadderOpeningHR
Workbook: SPSHRAtLadderOpening.xls

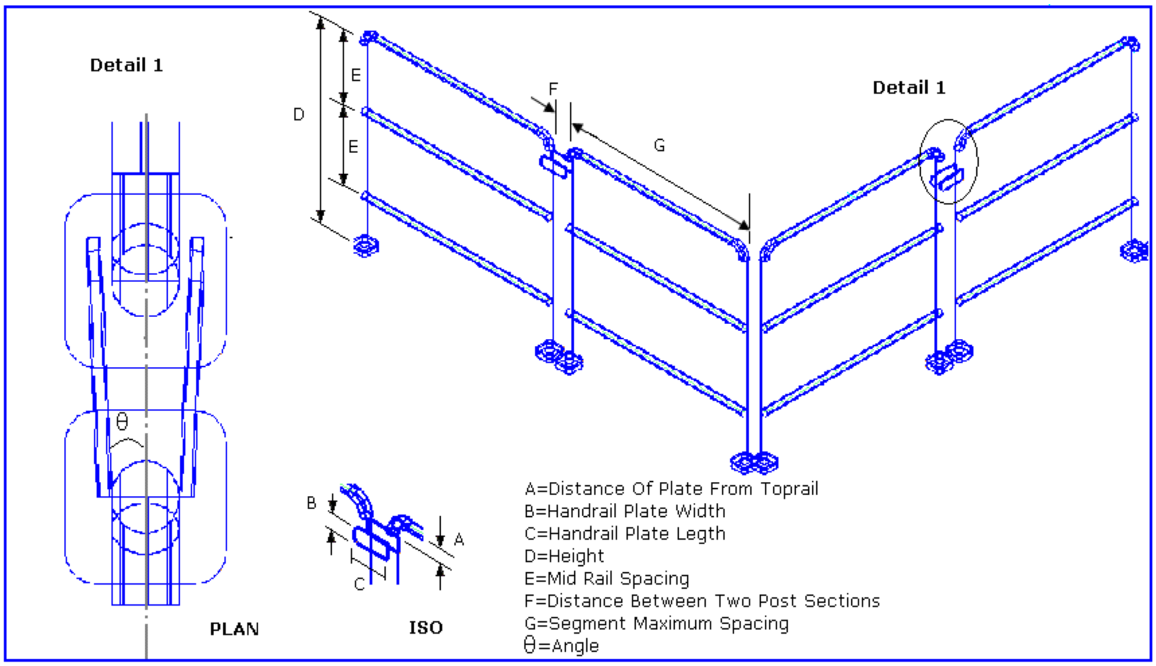
Workbook Sheet: HRAtLadderOpening
Prog ID: SPSLadderOpeningHR.LadderOpeningHR



SPSRemovableHR.RemovableHRTypeA

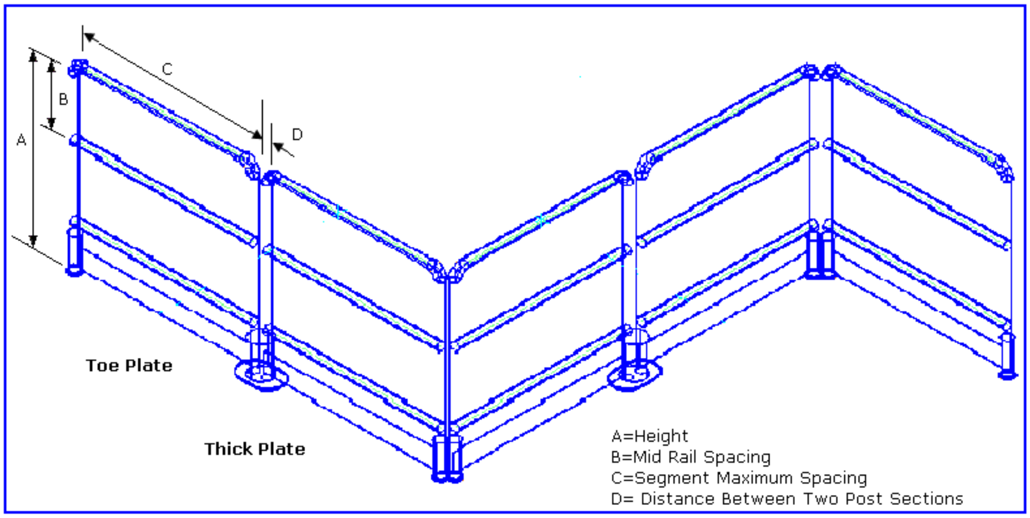
Description: removable handrail, type A
Symbol Name: SPSRemovableHR.RemovableHRTypeA
Workbook: SPSRemovableHandrails.xls

Workbook Sheet: Removable_HRTypeA
Prog ID: SPSRemovableHR.RemovableHRTypeA



SPSRemovableHR.RemovableHRTypeB

Description: removable handrail, type B
Symbol Name: SPSRemovableHR.RemovableHRTypeB
Workbook: SPSRemovableHandrails.xls
Workbook Sheet: Removable_HRTypeB
Prog ID: SPSRemovableHR.RemovableHRTypeB



SPSStormRail

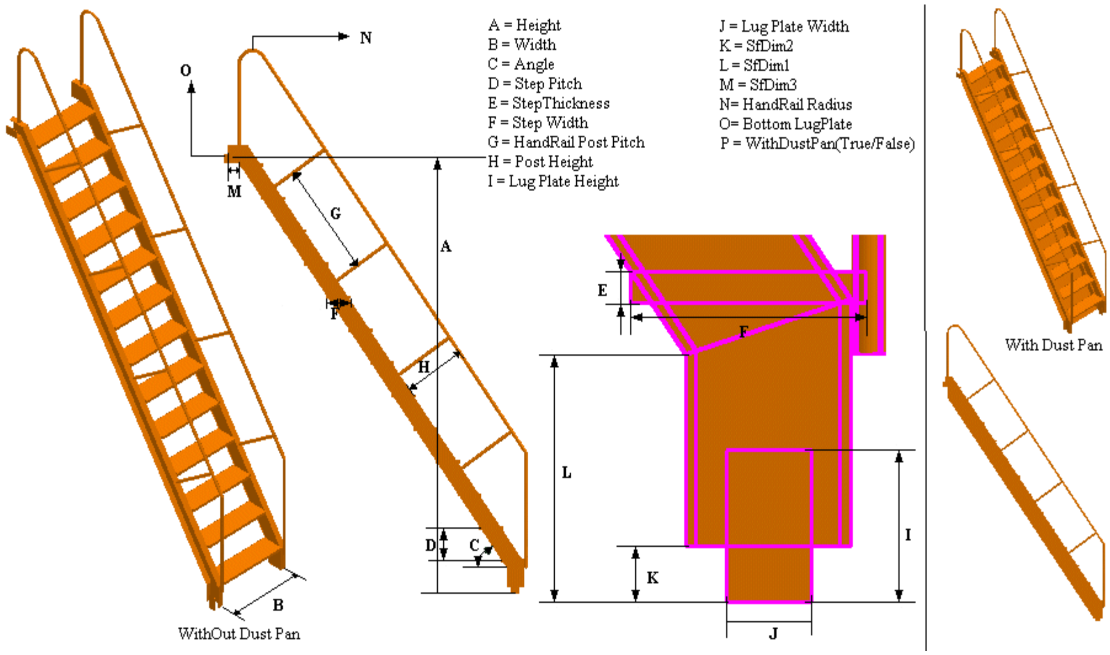
Description: storm railing
Symbol Name: SPSStormRailing.StormRailing
Workbook:
Workbook Sheet:
Prog ID: SPSStormRailing.StormRailing

Stairs and Ladders

Topics

SPSInclinedLadderMacros.InclLadderTypeA

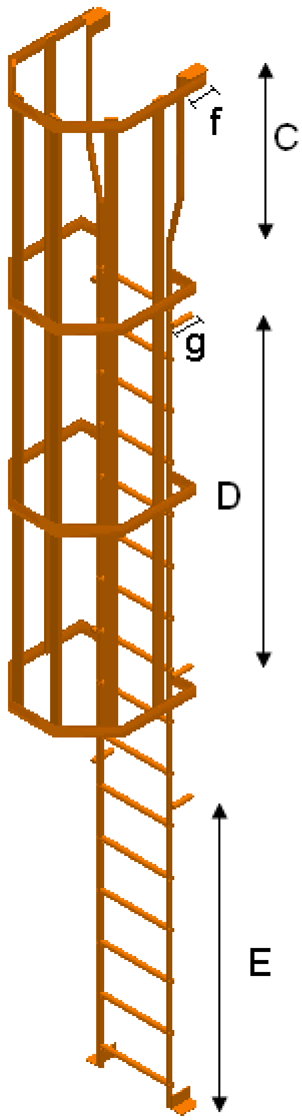
Description: inclined ladder, type A
Symbol Name: SPSInclinedLadderMacros.InclLadderTypeA
Workbook: StructStairsWithInclindedLadder.xls
Workbook Sheet: InclinedLadderTyA
Prog ID: SPSInclinedLadderMacros.InclLadderTypeA

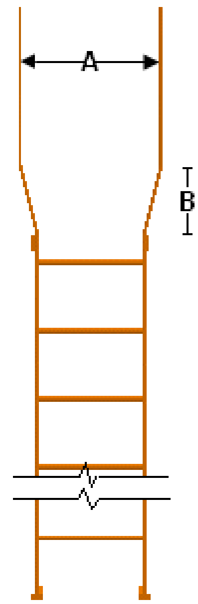


SPSLadderWithCage.LadderCageTypeA

Description: ladder with cage, type A
Symbol Name: SPSLadderWithCage.LadderCageTypeA
Workbook: SPSLadderWithCage.xls

Workbook Sheet: LadderWithCageTypeA5
Prog ID: SPSLadderWithCage.LadderCageTypeA





A-Width Above Tapper

B-Height of Tapper

C-Top Extension

D-Top Support Leg Pitch

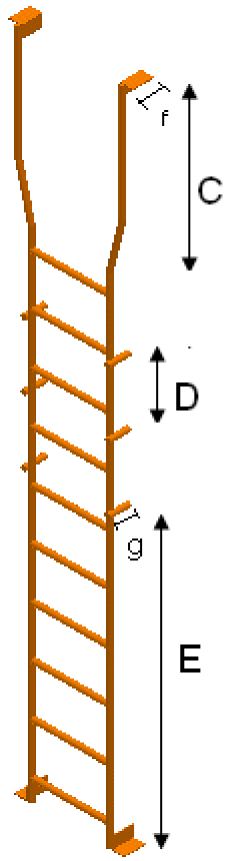
E-Bottom Support Leg Pitch

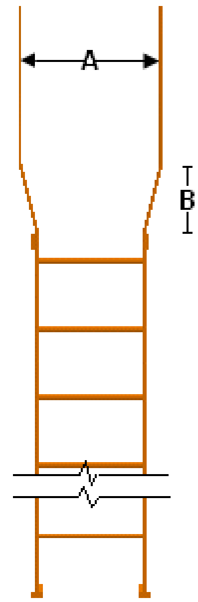
f-End connection Length

g-Support Leg Length

SPSLadderWithCage.LadderCageTypeB

Description: ladder with cage, type B
Symbol Name: SPSLadderWithCage.LadderCageTypeB
Workbook: SPSLadderWithCage.xls
Workbook Sheet: LadderWithCageTypeB5
Prog ID: SPSLadderWithCage.LadderCageTypeB





A-Width Above Tapper

B-Height of Tapper

C-Top Extension

D-Top Support Leg Pitch

E-Bottom Support Leg Pitch

f-End connection Length

g-Support Leg Length

SPSLadderMacros

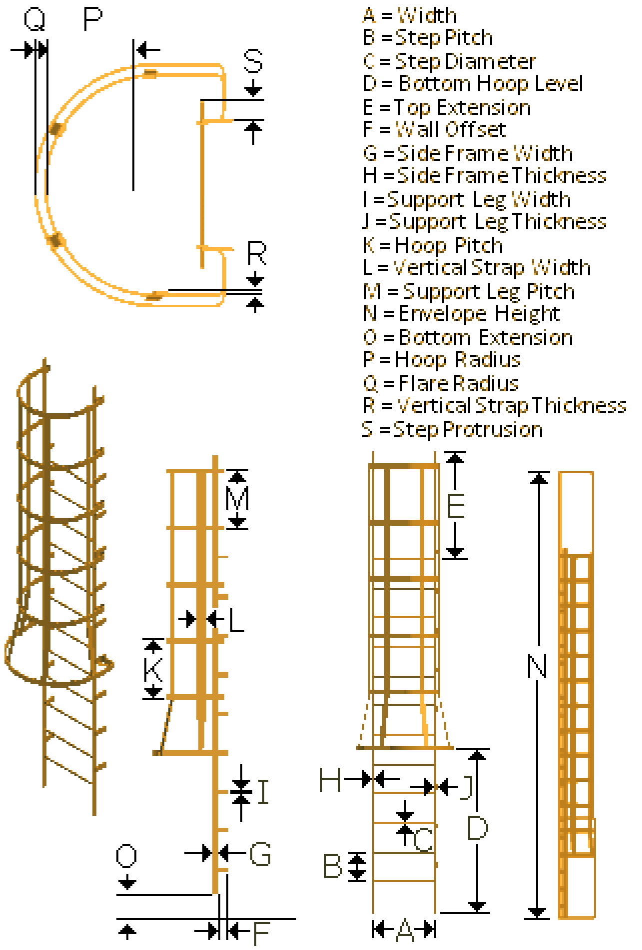
Description: ladder with safety cage
Symbol Name: SPSLadderMacros.LadderTypeA
Workbook: StructLadders.xls
Workbook Sheet: LadderTypeA
Inputs, Outputs, and Aspects:

Input Name = "Width"
Input Description = "Width"
Input Name = "Angle"
Input Description = "Angle"
Input Name = "StepPitch"
Input Description = "StepPitch"
Input Name = "SupportLegPitch"
Input Description = "SupportLegPitch"
Input Name = "SupportLegWidth"
Input Description = "SupportLegWidth"
Input Name = "SupportLegThickness"
Input Description = "SupportLegThickness"
Input Name = "SideFrameWidth"
Input Description = "SideFrameWidth"
Input Name = "SideFrameThickness"
Input Description = "SideFrameThickness"
Input Name = "StepDiameter"
Input Description = "StepDiameter"
Input Name = "VlDim1"
Input Description = "VlDim1"
Input Name = "VlDim2"
Input Description = "VlDim2"
Input Name = "VlDim3"
Input Description = "VlDim3"
Input Name = "WallOffset"
Input Description = "WallOffset"
Input Name = "Span"
Input Description = "Span"
Input Name = "Height"
Input Description = "Height"
Input Name = "Length"
Input Description = "Length"
Input Name = "WithWallSupports"
Input Description = "WithWallSupports"
Input Name = "NumSteps"
Input Description = "NumSteps"
Input Name = "StepProtrusion"
Input Description = "StepProtrusion"
Input Name = "WithSafetyHoop"
Input Description = "WithSafetyHoop"
Input Name = "HoopPitch"
Input Description = "HoopPitch"
Input Name = "BottomHoopLevel"
Input Description = "BottomHoopLevel"
Input Name = "HoopClearance"

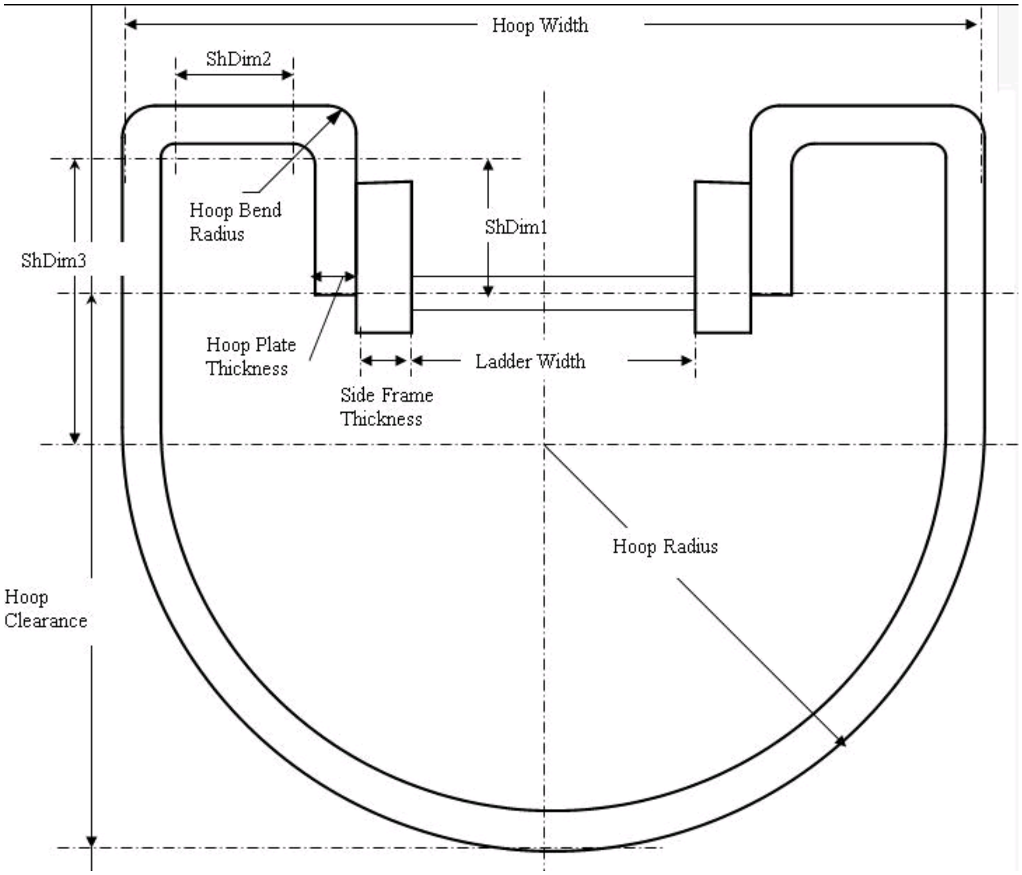
Input Description = "HoopClearance"
Input Name = "HoopRadius"
Input Description = "HoopRadius"
Input Name = "HoopPlateThickness"
Input Description = "HoopPlateThickness"
Input Name = "HoopPlateWidth"
Input Description = "HoopPlateWidth"
Input Name = "HoopBendRadius"
Input Description = "HoopBendRadius"
Input Name = "HoopOpening"
Input Description = "HoopOpening"
Input Name = "ShDim1"
Input Description = "ShDim1"
Input Name = "ShDim2"
Input Description = "ShDim2"
Input Name = "ShDim3"
Input Description = "ShDim3"
Input Name = "FlareClearance"
Input Description = "FlareClearance"
Input Name = "FlareRadius"
Input Description = "FlareRadius"
Input Name = "HoopFlareBendRadius"
Input Description = "HoopFlareBendRadius"
Input Name = "FlareShDim1"
Input Description = "FlareShDim1"
Input Name = "FlareShDim2"
Input Description = "FlareShDim2"
Input Name = "FlareShDim3"
Input Description = "FlareShDim3"
Input Name = "HoopFlareHeight"
Input Description = "HoopFlareHeight"
Input Name = "HoopFlareMaxHeight"
Input Description = "HoopFlareMaxHeight"
Input Name = "VerticalStrapWidth"
Input Description = "VerticalStrapWidth"
Input Name = "VerticalStrapThickness"
Input Description = "VerticalStrapThickness"
Input Name = "VerticalStrapCount"
Input Description = "VerticalStrapCount"
Input Name = "TopExtension"
Input Description = "TopExtension"
Input Name = "BottomExtension"
Input Description = "BottomExtension"
Input Name = "Justification"
Input Description = "Justification"
Input Name = "TopSupportSide"
Input Description = "TopSupportSide"
Input Name = "IsAssembly"
Input Description = "IsAssembly"
Input Name = "EnvelopeHeight"
Input Description = "EnvelopeHeight"
Input Name = "Primary_SPSMaterial"
Input Description = "Primary_SPSMaterial"

Input Name = "Primary_SPSGrade"
Input Description = "Primary_SPSGrade"
Aspect Name = "Physical"
Aspect Description = "Physical representation"
Output Name = "LeftSideFrame1"
Output Description = "Left side frame element"
Output Name = "RightSideFrame1"
Output Description = "Right side frame element"
Aspect Name = "DetailPhysical"
Aspect Description = "DetailPhysical representation"
Aspect Name = "OperationalSub"
Aspect Description = "Operational representation"
Output Name = "OperationalEnvelope1"
Output Description = "Operational Envelope of the Vertical Ladder"
Output Name = "Step"
Output Description = "Ladder Step"
Output Name = "LeftSupportLeg"
Output Description = "Left Support Leg"
Output Name = "RightSupportLeg"
Output Description = "Right Support Leg"
Output Name = "SafetyHoop"
Output Description = "SafetyHoop"

Output Name = "SafetyHoop"
Output Description = "SafetyHoop"



A = Width
B = Step Pitch
C = Step Diameter
D = Bottom Hoop Level
E = Top Extension
F = Wall Offset
G = Side Frame Width
H = Side Frame Thickness
I = Support Leg Width
J = Support Leg Thickness
K = Hoop Pitch
L = Vertical Strap Width
M = Support Leg Pitch
N = Envelope Height
O = Bottom Extension
P = Hoop Radius
Q = Flare Radius
R = Vertical Strap Thickness
S = Step Protrusion



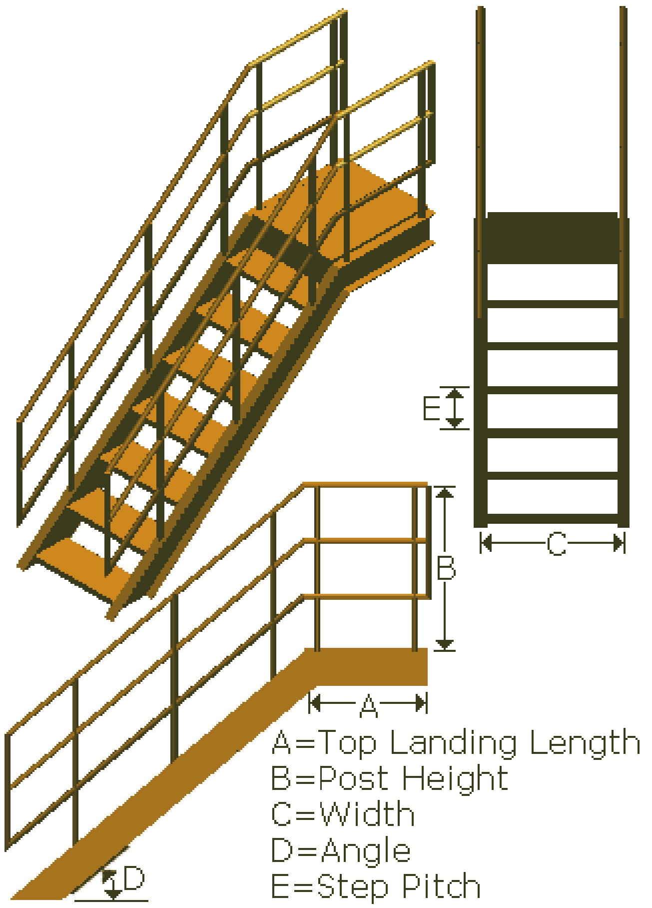
SPSStairMacros

Description: stairs with side handrails and optional top landing
Symbol Name: SPSStairMacros.StairTypeA
Workbook: StructStairs.xls
Workbook Sheet: StairTypeA
Inputs, Outputs, and Aspects:

Input name = "Width"
Input Description = "Width"
Input name = "Angle"
Input Description = "Angle"
Input name = "StepPitch"
Input Description = "StepPitch"
Input name = "Height"
Input Description = "Height"
Input name = "NumSteps"
Input Description = "NumSteps"
Input name = "Span"
Input Description = "Span"
Input name = "Length"
Input Description = "Length"

Input name = "Justification"
Input Description = "Justification"
Input name = "TopSupportSide"
Input Description = "TopSupportSide"
Input name = "SideFrame_SPSSectionName"
Input Description = "SideFrame_SPSSectionName"
Input name = "SideFrame_SPSSectionRefStandard"
Input Description = "SideFrame_SPSSectionRefStandard"
Input name = "HandRail_SPSSectionName"
Input Description = "HandRail_SPSSectionName"
Input name = "HandRail_SPSSectionRefStandard"
Input Description = "HandRail_SPSSectionRefStandard"
Input name = "Step_SPSSectionName"
Input Description = "Step_SPSSectionName"
Input name = "Step_SPSSectionRefStandard"
Input Description = "Step_SPSSectionRefStandard"
Input name = "SideFrameSectionCP"
Input Description = "SideFrameSectionCP"
Input name = "SideFrameSectionAngle"
Input Description = "SideFrameSectionAngle"
Input name = "HandRailSectionCP"
Input Description = "HandRailSectionCP"
Input name = "HandRailSectionAngle"
Input Description = "HandRailSectionAngle"
Input name = "StepSectionCP"
Input Description = "StepSectionCP"
Input name = "StepSectionAngle"
Input Description = "StepSectionAngle"
Input name = "Primary_SPSMaterial"
Input Description = "Primary_SPSMaterial"
Input name = "Primary_SPSGrade"
Input Description = "Primary_SPSGrade"
Input name = "PlatformThickness"
Input Description = "PlatformThickness"
Input name = "WithTopLanding"
Input Description = "WithTopLanding"
Input name = "TopLandingLength"
Input Description = "TopLandingLength"
Input name = "PostHeight"
Input Description = "PostHeight"
Input name = "HandRailPostPitch"
Input Description = "HandRailPostPitch"
Input name = "NumMidRails"
Input Description = "NumMidRails"
Input name = "IsAssembly"
Input Description = "IsAssembly"
Input name = "IsSystem"
Input Description = "IsSystem"
Input name = "EnvelopeHeight"
Input Description = "EnvelopeHeight"
Aspect name = "Physical"
Aspect Description = "Physical representation"
Output name = "LeftSideFrame"

Output Description = "Left side frame element"
Output name = "RightSideFrame"
Output Description = "Right side frame element"
Aspect name = "DetailPhysical"
Aspect Description = "DetailPhysical representation"
Aspect name = "OperationalSub"
Aspect Description = "Operational representation"
Output name = "OperationalEnvelope1"
Output Description = "Operational Envelope of the Stair"



Index

P

R

S

T

U

V

W